Computing Parameter Sloppiness in Nondeterministic Economic Models



William Scarrold
Wolfson College
University of Oxford

A thesis submitted for
Doctor of Philosophy
Trinity 2024

Abstract

Various models have been proposed and implemented to simulate economic systems, including general equilibrium models and agent-based models. Analysing and understanding these models, however, is limited by their complexity; particularly in the case of agent-based models, describing their full range of behaviours and the sensitivities of such behaviours to changes in parameters can prove intractable. This problem has motivated previous approaches to efficiently exploring parameter spaces using Fisher information matrix-like objects. Such methods appeal to the notion of "sloppiness", a form of parameter sensitivity analysis, and generally rely on computing Fisher information matrices using numerical finite differences. While frequently successful, this approach struggles when the model of interest exhibits stochasticity. Stochastic finite difference methods must then be invoked, requiring not only many model runs at each point of interest but also checks on the convergence of the computed stochastic derivatives. This thesis proposes several methods to compute model sloppiness without stochastic finite differencing. These methods include (1) analysing the impulse response functions that characterise macroeconomic general equilibrium models, (2) creating differentiable surrogate models with deterministic neural networks, and (3) using generative autoencoder networks capable of simulating the probability distribution of a model's outputs. We apply these techniques to general equilibrium and agent-based models, finding each is able to accurately identify the stiff and sloppy directions. We also demonstrate how such techniques can be used to analyse several widely-used economic models.

Acknowledgements

It would not have been possible to do this DPhil without the support of many people.

First and foremost, I would like to thank my supervisor, Doyne Farmer, for his guidance and support throughout my DPhil journey. Thank you for being a steadfast presence throughout my time at Oxford, and for being an inspiration in so many ways.

I greatly appreciate all the time I have spent with members of INET over the years, both academically and socially. I would especially like to thank Blas Kolic, François Lafond, Joel Dyer, José Moran, and Maria del Rio Chanona, who have all contributed greatly to my experience at Oxford. I also thank Henry Macdonald for the support he has given me throughout my DPhil, including hours of chat on academic (and non-academic) topics.

Thank you to Dorothy Nicholas for the immense support she has provided to me, and all members of my research group. Dorothy, it is safe to say every DPhil student in the INET complexity economics group owes so much to you for your help throughout our time studying at Oxford. Your dedication and kindness have always been an enormous help.

I would like to extend my heartfelt thanks to my family. To Mum, Dad, Louis, Mary, and Alastair, thank you for your constant encouragement and for all the time you have spent supporting me throughout my journey.

To my daughter, Olivia, thank you for joining us in this world, and bringing joy and wonder to all those around you. I look forward to sharing the wonders of mathematics and economics with you, if it is of any interest.

Most importantly, I wish to thank my partner, Clare, for her endless sacrifice, without which this DPhil would not be possible. Clare, I cannot express my gratitude enough, and I just hope I can make it up to you over the coming decades.

Contents

Introduction **1**

0.1 Overview of Macroeconomic Modelling 2
Goal 1: Forecasting 3
Goal 2: Policy Analysis 3
Model Types 3
0.2 Model Sloppiness 6
0.3 Sloppiness vs Parameter Irrelevance 8
0.3.1 Illustrative Example 9
0.4 Deriving Model Sloppiness 10
0.4.1 Least-Squares Fitting 10
0.4.2 Information Geometry 12
Fisher Information Matrix 13
Kullback–Leibler Divergence and the FIM 14
0.4.3 Non-Gaussian Cases 15
0.4.4 Sloppiness in Deterministic Models 15
0.4.5 Computing Sloppiness 16
Output Normalisation 16
Parameter Normalisation 17
0.5 Behaviour Space, and the Meaning of Parameter Sloppiness 17
0.5.1 Behaviour Space and Model Manifolds 18
0.5.2 Moving Within the Model Manifold 21
Interpreting $J^T J$ 21
0.5.3 Sloppy Subspaces 22
0.6 Challenges Resulting from Sloppiness 23
0.6.1 Other Methods for Reducing Parameter Space 23
Regularisation and Model Selection Techniques 23
Principal Component Analysis 24
0.7 Motivating Problems for this Thesis 24

0.7.1 Exploration of Parameter Space . . . 26
0.8 Conclusion . . . 27

1 Sloppiness in DSGE Models **28**
1.1 Literature Review: DSGE Models . . . 28
1.1.1 History . . . 28
Controversy Over DSGEs . . . 30
1.1.2 DSGE Structure/Design . . . 31
Households . . . 32
Firms . . . 32
Markets . . . 32
Central Bank . . . 32
Random Shocks . . . 32
1.1.3 Steady States and Linearisation . . . 33
1.1.4 Impulse Response Functions . . . 34
DSGE and IRF Definitions . . . 34
1.1.5 Computational Expense: Generating Time Series vs Computing Impulse Responses . . . 36
A Note on Steady States and RNG Seeds . . . 38
1.2 Method . . . 39
1.2.1 Choosing IRF Variables and Shocks . . . 39
1.2.2 Performance Measurement . . . 40
Eigenvalue Spectra . . . 41
Cosine Similarity Matrix . . . 41
1.2.3 First Test Case: Smets & Wouters Model . . . 43
1.2.4 Other DSGE Test Cases . . . 43
1.3 Results . . . 44
1.3.1 SW Eigenvalue Spectra . . . 45
1.3.2 SW Cosine Similarity . . . 45
1.3.3 Measuring Deviation . . . 46
1.4 Results for Other DSGE Models . . . 47
1.4.1 Cosine Similarity Matrices and Eigenvalue Spectra . . . 51
1.5 Discussion . . . 52
1.6 Chapter 1 Conclusion . . . 53

2 Surrogate Modelling: Mimicking Complex Systems with Neural Networks 54
2.1 Literature Review: Surrogate Modelling . . . 55
2.1.1 History . . . 55
2.1.2 Challenges in Surrogate Modelling . . . 56
2.2 Defining a Surrogate . . . 57
2.2.1 Surrogate Methods . . . 58
2.2.2 Sampling Plan . . . 59
2.3 Neural Network Architecture . . . 60
2.3.1 Fully Connected FNNs . . . 60
Independent and Dependant Variables . . . 62
2.4 Our Surrogate Method . . . 62
2.4.1 Parameter Normalisation . . . 63
2.5 Model Features to Emulate . . . 63
2.5.1 Neural Network Candidates . . . 65
2.6 Models . . . 66
2.6.1 A Toy Function, $\phi(\theta, t)$ . . . 67
2.6.2 Hodgkin-Huxley Model . . . 68
2.6.3 An Agent-based SIR Model . . . 70
2.6.4 Brock-Hommes Model . . . 72
2.6.5 An Agent-Based Ramsey–Cass–Koopmans Model . . . 74
2.6.6 Mark-0 Agent-Based Model . . . 75
2.7 Results . . . 76
2.7.1 Toy Model $\phi(\theta, t)$ Results . . . 76
2.7.2 HH Results . . . 79
2.7.3 Agent-based SIR Results . . . 81
2.7.4 BH Results . . . 83
2.7.5 Agent-based RCK Results . . . 85
2.7.6 Mark-0 Results . . . 87
2.7.7 Eigenvalue Spectra – All Models . . . 89
2.7.8 RMSE Values – All Models . . . 91
2.8 Discussion . . . 91
2.8.1 Time Cost of Eigenvector Computation . . . 92
2.8.2 Difficulty Identifying the Sloppiest Eigendirections . . . 93
SFD in Presence of Gaussian Noise . . . 93
Surrogates and Sloppy Eigendirections . . . 94

2.8.3 Consistent Sensitivities, Inconsistent Simulations 95
2.8.4 Exploring Large Parameter Spaces Using Surrogates 96
2.8.5 Alternative Architectures: Recurrent Neural Networks, LSTMs, and GRUs 96
2.9 Chapter 2 Conclusion 97

3 Generative Neural Networks as Surrogates 98
3.1 Generative Models 101
3.2 Literature Review: Autoencoders 102
3.2.1 Basic Structure 102
3.2.2 History of Autoencoders 103
3.2.3 Autoencoder Architectures 104
Simple Autoencoder Structure 104
VAE Structure 105
CVAE Structure 106
3.2.4 Existing Applications of CVAE Networks 106
CVAEs as Surrogates 106
3.2.5 Other Generative Methods 107
3.3 Our Generative Surrogate Method 108
3.4 Results 110
3.4.1 HH Results 110
3.4.2 Agent-based SIR Results 111
3.4.3 BH ABM Results 112
3.4.4 Agent-based RCK Results 114
3.4.5 Mark-0 Results 115
3.4.6 RMSE Values – All Models 116
3.4.7 Speed comparison – All Models 117
3.4.8 The Eigenvector-Swapping Problem and Convolution 118
Matrix Convolution 119
Convolution Result (Mark-0) 120
3.5 Discussion 122
3.5.1 Predicting the need for Nondeterministic Surrogates 123
3.5.2 Beyond Sloppiness: Generative Surrogates for Efficient Analysis and Decision Support 123
3.6 Chapter 3 Conclusion 124

Conclusion 125

A **128**
A.1 FIM for a Multivariate Gaussian Distribution 128
A.2 KL Divergence 129

B **131**
B.1 DSGE Model Details 131
B.1.1 Smets & Wouters model 131
B.1.2 Aguiar & Gopinath VAR model 133
B.1.3 Gali 2015 (Chapter 8) model 134
B.1.4 Schmitt-Grohe & Uribe model 135
B.1.5 An RBC model 136
B.1.6 Hansen model 136

C **138**
C.1 Model Parameters, Outputs, and Initial Conditions 138
C.1.1 Toy model $\phi(\theta, t)$ 138
C.1.2 Hodgkin-Huxley Model 139
C.1.3 Agent-based SIR 139
C.1.4 Brock-Hommes 140
C.1.5 RCK ABM 140
C.1.6 Mark 0 141
C.2 Deterministic Surrogate Network Specifications 142

D **144**
D.1 Generative Surrogate Network Specifications 144
D.1.1 Hodgkin-Huxley Model 144
D.1.2 Agent-based SIR 145
D.1.3 Brock-Hommes 145
D.1.4 RCK ABM 146
D.1.5 Mark 0 146

Bibliography **148**

List of Figures

1 Parameter space and behaviour space, for exponential function. . . . 20

1.1 Simulation of Smets & Wouters DSGE model. 33
1.2 Impulse response functions of a DSGE model. 36
1.3 Mean cosine similarity of random vectors. 42
1.4 SW eigenvalue spectrum. 45
1.5 SW cosine similarity matrix. 46
1.6 Eigenvalue spectra for DSGE models. 48
1.7 Cosine similarity matrices for DSGE models. 49
1.8 Deviation plots for DSGE time series simulations. 50

2.1 Feedforward neural network structure. 61
2.2 Procedure for constructing an artificial neural network surrogate. . . 65
2.3 Simulations of toy model, $\phi(\theta, t)$. 67
2.4 Realisations of the Hodgkin-Huxley model. 69
2.5 Construction of a Watts-Strogatz network. 71
2.6 Realisations of the agent-based SIR model. 72
2.7 Realisations of the Brock-Hommes model. 73
2.8 Realisations of the agent-based RCK model. 75
2.9 Realisations of the Mark-0 model. 76
2.10 Behaviour of the deterministic surrogate for $\phi(\theta, t)$. 77
2.11 $\phi(\theta, t)$ cosine similarity and eigenvalue spectrum. 78
2.12 Behaviour of the deterministic surrogate for the Hodgkin-Huxley model. 79
2.13 Hodgkin-Huxley cosine similarity and eigenvalue spectrum. 80
2.14 Behaviour of the deterministic surrogate for the agent-based SIR model. 81
2.15 Agent-based SIR model cosine similarity and eigenvalue spectrum. . . 82
2.16 Behaviour of the deterministic surrogate for the Brock-Hommes model. 83
2.17 Brock-Hommes cosine similarity and eigenvalue spectrum. 84
2.18 Behaviour of the deterministic surrogate for the Agent-based RCK model. 85

2.19 Agent-based RCK cosine similarity and eigenvalue spectrum. 86
2.20 Realisations of the Mark-0 model, along with runs from its deterministic surrogate. 87
2.21 Mark-0 cosine similarity and eigenvalue spectrum. 88
2.22 Eigenvalue spectrum plots for all models and their surrogates. 90

3.1 Runs of $H(\theta, t)$, with $\varepsilon_0, \varepsilon_1$ held constant. 100
3.2 Runs of $H(\theta, t)$, showing zero mean. 100
3.3 Realisations of $H(\theta, t)$, along with runs from its deterministic surrogate. 101
3.4 Structure of a simple autoencoder network. 103
3.5 Structure of simple autoencoder, VAE, and CVAE networks. 105

C.1 Structure of the FNN surrogate for the agent-based SIR model. . . . 143

D.1 Structure of the CVAE surrogate for the agent-based SIR model. . . . 147

Introduction

Modelling variability in economic systems dates back to Adam Smith's *The Wealth of Nations* [137], in which the concept of the "Invisible Hand" was used to describe the dynamics of production and pricing in fluctuating environments. In the early 20th century, further work allowed this variability to be treated probabilistically, including notable contributions from John Maynard Keynes and Frank Knight [84, 88]. Since then, multiple sources of randomness in modern economic systems have been identified and diversely incorporated into economic models. In many "mainstream" economic models – including widely used general equilibrium models used to simulate whole economies – randomness stems from exogenous, unanticipated "shocks" [125, 136, 36]. Examples of such shocks include unexpected increases (or decreases) in technological progress, monetary policy shocks, and changes in individuals' preferences. In contrast, other models – including agent-based models (ABMs) – propose that fluctuations can be induced by endogenous processes [7, 10]. Such models do not necessarily assume a single equilibrium exists, and even if one does exist, it is not assumed feasible for the system to find and maintain that equilibrium.

Both in terms of their ability to replicate observed features in the economy, and their ability to accurately forecast the future economic states, economic models benefit from sensitivity analysis regarding model parameters in the assessment of model suitability. This type of assessment is helpful for several reasons, including identifying important (and unimportant) parameters, which can aid experimental design, and guide policy recommendations when some parameters are controllable [121]. Additionally, uncovering the full range of behaviours of an economic model can help modellers understand its ability to replicate observed economic phenomena, and assist policymakers in identifying changes in the economy (e.g., financial crises) that may result if parameters drift in certain directions [112].

However, the nondeterministic nature of these models creates challenges when carrying out sensitivity analysis. This thesis attempts to tackle this central problem, using the concept of model "sloppiness".

In Section 0.1 of this thesis, we first discuss the general motivations for developing economic models. We then present the technical preliminaries necessary for understanding the mathematics behind model sloppiness analysis. This is followed by a discussion of sloppiness analysis in detail, including how it is used to study parameter sensitivity. We also describe the history of sloppiness analysis, and existing applications. This chapter provides context for the subsequent chapters of the thesis, which present three novel methods of computing sloppiness in nondeterministic models.

In Chapter 1, we present a method for computing sloppiness in dynamic stochastic general equilibrium models (DSGEs). This utilises a feature that is "internal" to many DSGEs: the set of impulse response functions (IRFs). The IRFs examine how a DSGE model behaves after a single shock, and we calculate IRF sensitivity to parameter changes in order to infer the sensitivity of the full system (which is more complex than the case of a single IRF because the system must constantly respond to new shocks as they occur). The methods developed in this chapter can be applied to DSGEs and vector autoregressions (VARs).

Chapter 2 of this thesis approaches the problem of sloppiness computation from the "outside", rather than the inside. The model of interest is run many times to generate a training set, which is then used to create "surrogates" that mimic the observed behaviour of the model. In this part, the economic model is treated as a "black box", in the sense that we do not probe its inner workings, but instead concern ourselves only with the behaviours observed when the model runs. We build our surrogates using artificial neural networks (ANNs), which are sufficiently flexible to capture the complex behaviour of a range of models. Additionally, our ANNs are differentiable by the backpropagation technique, making the computations of derivatives (and hence model sloppiness) extremely efficient.

Finally, in Chapter 3 we present an alternative method of building differentiable ANN surrogates. We create stochastic, "generative" autoencoder networks that can emulate the nondeterminism present in many economic models. These generative surrogates can take a single parameter vector as an input, and generate many outputs that collectively have the same probability distribution as the outputs of a nondeterministic model of interest.

0.1 Overview of Macroeconomic Modelling

This section discusses goals of economic modelling and the main modelling techniques of the past and present. Accurate economic models are desirable for many reasons,

including their role in forecasting and policy analysis.

Goal 1: Forecasting

Accurately predicting both short- and long-term trajectories of the economy allows central banks to make policy decisions [20], governments to forecast budget deficits [159], and individuals and firms to make expenditure and investment decisions with greater certainty. For example, individuals may be more willing to invest in housing if forecasts suggest near-term interest rate rises are unlikely [76], and banks may be less willing to lend to firms if a recession is anticipated [22]. Long-term forecasting can include predicting key trends in macroeconomic variables, but can also cover other topics, for example the future nature of employment [131] (including which occupations may be eliminated due to artificial intelligence and how soon) and the environmental sustainability of current economic trajectories [37].

Goal 2: Policy Analysis

The impacts of economic policies can be extensive, and often involve trade-offs between positive and negative consequences. For example, there is active debate regarding the trade-off between economic growth and inequality (the existence of which is itself disputed [113, 16]), and models that accurately predict risks involved in financial (de)regulation are important for avoiding and mitigating systemic crises [144] (such as a repeat of the Global Financial Crisis). Further examples include strategic decision-making around long-term investments (e.g., the extent to which governments should invest in technological research and development [99]), the need to create alternative training and employment pathways in the face of technological change [131], and how to efficiently mitigate the impact of economic activity on the environment [37].

Model Types

Mathematical economic models are well over a century old. Creating a complete taxonomy of all existing modelling techniques would be a near-impossible exercise, but many of the most significant fall into one of a few categories [72]. These include simple statistical approaches such as regression analysis, correlation analysis, and simultaneous equation modelling. Additionally, there are Keynesian approaches, including the IS-LM mechanism which links interest rates and GDP in a macroeconomy [73]. Other notable types include real business cycle (RBC) models, DSGE models, advanced statistical approaches (such as error correction models [12]), and ABMs.

Each of these techniques has advantages and disadvantages. Some of these models contain very strong and unrealistic assumptions (e.g., perfectly rational behaviour is often assumed in DSGEs [129, 153]), while others may make inappropriate statistical assumptions [72]. Additionally, some models can be computationally challenging to simulate. This last property is often true of ABMs [4, 104, 111]. This is discussed below in more detail, for each model type.

Simple Statistical Models

A simple VAR model is capable of simulating a multivariate time series with interdependent variables. Here we will describe a VAR model's structure, its strengths and weaknesses as an economic model, and other statistical models of similar complexity. In a simple VAR model, the value of each variable at time t is assumed to be a linear function of all variables at time $t-1$. That is,

$$Y_t = C + AY_{t-1} + \varepsilon_t, \tag{0.1}$$

where Y_t is a vector of the variables (e.g., GDP, inflation, interest rates) at time t, C is a vector of constants (representing intercepts), A is a matrix of coefficients, and ε_t is a vector of homoscedastic exogenous shocks. A two-variable VAR has the form:

$$\begin{bmatrix} y_{1,t} \\ y_{2,t} \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} + \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} y_{1,t-1} \\ y_{2,t-1} \end{bmatrix} + \begin{bmatrix} \varepsilon_{1,t} \\ \varepsilon_{2,t} \end{bmatrix}. \tag{0.2}$$

An advantage of using VAR models is the relative ease of estimating the model's parameters (a_{ij}), which involves matrix inversion and multiplication [30]. Although matrix inversion can be time consuming, this is usually not nearly as computationally expensive as the Bayesian Monte Carlo methods used in modern DSGE and ABM parameter estimation procedures [5]. The parameters of a VAR model are also interpretable, as a given parameter, a_{ij}, represents the effect variable y_j at time $t-1$ has on variable y_i at time t.

Disadvantages of VAR models include the strong statistical assumptions they require [30], including the requirement that the variables are stationary, and that relationships between variables do not vary when policy changes are made (discussed further in section 1.1.1). Additionally, economically meaningful restrictions regarding the relationships between variables cannot be imposed, such as restricting the model from allowing one variable to have an immediate effect on another [133].

More complex statistical models have been applied to economic modelling, including Simultaneous Equation Models (SEMs) and Structural VARs (SVARs), which allow economically meaningful parameter restrictions to be imposed [24]. Error Correction Models (ECMs) can allow a modeller to relax some statistical assumptions [45].

The strengths of these models are similar to those of VARs (including interpretability and relative ease of estimation), but they still make unrealistic assumptions, including that parameter values do not change over time.

DSGE Models

DSGE models were created to address criticisms of the simpler statistical approaches, which were dominant in the immediate post-war era. The major components of a DSGE model are equations representing the relationships between demand and supply across multiple markets, and equations representing households' and firms' decisions regarding levels of demand and price setting. Households and firms are assumed to be rational [129, 153], in the sense that they dynamically vary their demand level, savings rate, and hours of work in such a way that their expected utility is maximised. Equations have terms representing unexpected, exogenous "shocks", which are the cause of variability in the economy (which would otherwise remain in an equilibrium steady state indefinitely).

The equations represent economically meaningful relationships between variables, which addresses a major criticism of earlier statistical approaches. Additionally, DSGEs can be "microfounded" [152], in the sense that the model's rules can be derived from assumed economic behaviour of individuals (rather than setting rules that only consider macroeconomic aggregates, such as GDP, which was standard in simple statistical approaches). While the estimation of parameters is more complex for DSGEs compared to VARs, advances in computational power have made the task of estimation (often using Bayesian methods) tractable [5].

However, DSGE models do share some of the weaknesses of VAR models and similar statistical approaches, including many of the same statistical assumptions [31]. Additionally, the assumption that individuals and firms are capable of forming rational beliefs (and act rationally) has been widely criticised as unrealistic. The absence of financial markets in most DSGE models [25] has been criticised, especially following the Global Financial Crisis (GFC), which was not predicted by widely-used DSGE models.

Section 1.1 discusses DSGE models in more depth, including their historical motivation, structure, and the controversy that has emerged regarding their use.

Agent-Based Models

ABMs have been developed to model the behaviour of a diverse range of systems. These include ecological systems, epidemiology, and social networks and other

complex systems [104]. ABMs have also been created to study economic systems, including financial markets and macroeconomic systems.

The rules of an ABM can be very flexible, allowing a broad range of behavioural rules, nonlinear interactions, and adaptive learning mechanisms [104, 4]. Like DSGEs, the models can be based on meaningful behavioural rules, but these rules are much more flexible in ABMs. Modellers can allow agent behaviours to vary dynamically based on heuristics, rather than the strong assumption that agents act rationally. Additionally, agent heterogeneity can be incorporated, making ABMs more realistic representations of the systems they model [104].

However, ABMs do have weaknesses, including the difficulty in choosing behavioural rules; there is only one way to be rational, but many ways to be irrational. Simulating ABMs can also be computationally expensive, which can make them difficult to apply to problems, especially if the system contains a large number of agents [104]. While advances in computational power have made simulation feasible, parameter estimation is more difficult. As a result of this difficulty, several methods have been created to estimate ABM parameters indirectly, including Bayesian Simulation-Based Inference [81, 44].

This thesis takes a relatively agnostic view on the suitability of different approaches to the problem of economic modelling. However, we tend to focus on ABM and DSGE models due to their current prominence in economic modelling communities. Large-scale macroeconomic ABMs are a relatively new approach to modelling entire economies, with some models showing promising early results [118]. Thus, developing methods that ABM modellers can apply is valuable. Simultaneously, the ongoing use of DSGEs by many large economic institutions [20] suggests that new techniques for analysis of these models could find widespread application.

0.2 Model Sloppiness

The concept of model sloppiness can be useful for applying and interpreting models with many parameters. For a given model, it is frequently observed that only a limited number of "stiff" directions in the parameter domain substantially affect the observable dynamics [121]. Simultaneously, there may exist several "sloppy" directions which have very little effect on model behaviour.

The theory of model sloppiness has been developed over more than two decades [27], and applied to mathematical models of various complex systems. These in-

clude models in systems biology [68], neuroscience [60], economics [112, 96], quantum physics [155], and engineering [121].

The derivation of stiff and sloppy directions has several immediate use cases:

1. **Sloppy** directions can be used to simplify models, make them more tractable, and fit them more easily to available data.

2. **Stiff** directions can be used to establish which parameters have the most influence on a model's behaviour. Identifying stiff directions allows one to explore parameter space efficiently, in order to identify the different behaviours (phases) a model is capable of producing. This is especially useful when it is difficult, a priori, to know all the possible behaviours of the model. For a many-parameter agent-based model with a large phase space, this can prove invaluable.

Examples of the former include Transtrum and Peng's 2014 [151] work, which introduces an algorithm for reducing the dimensionality of models – the "Manifold Boundary Approximation Method" (MBAM). This technique simplifies complex models by following sloppy directions, but considers movements not through parameter space, but instead through a "behaviour space", which represents the possible outputs of the model. The possible behaviours (i.e., outputs) of the model make up a subspace of behaviour space, named the "model manifold". As discussed further in Section 0.5, the structure of model manifolds is closely linked to sloppiness in a model, and it allows one to gain a parameter-independent understanding of model sloppiness [121]. MBAM has been successfully applied a range of fields, including information and communication technology [145] and biology [156]. However, it does require model behaviour to vary smoothly [121], and is only a local method of model reduction, meaning it may fail to discover some possible simplifications.

In Naumann-Woleske et al.'s 2021 [112] work, several economic models – including DSGEs and ABMs – are shown to have a sloppy structure. A novel algorithm is also introduced, which pays particular attention to the identification of stiff directions. This is done in order to explore parameter space and discover the parameter variations that lead to large changes in model behaviour. The algorithm explores parameter space from some initial point θ, by taking incremental steps in either the stiffest (v_1) or second-stiffest eigenvector (v_2), as determined by a random draw. The scalars:

$$p_1 = \frac{\lambda_1}{\lambda_1 + \lambda_2} \tag{0.3}$$

and

$$p_2 = \frac{\lambda_2}{\lambda_1 + \lambda_2} \tag{0.4}$$

are the probabilities of following the stiffest or second-stiffest directions, respectively. The scalars λ_1 and λ_2 are the eigenvalues associated with v_1 and v_2, respectively. After each step, the matrix $J^T J$ must be recomputed, from which updated values for v_1 and v_2 are derived. The need to compute $J^T J$ and its eigenvectors at many points in parameter space may present a significant computational challenge, depending on the model being investigated.

In addition to the applications discussed above, determining both sloppy and stiff directions can be useful for experimental design [6]. This is especially valuable when data may be expensive to generate (e.g., biological experiments can be costly and time-consuming, as can the collection of economic data through surveys and the processing of large and unclean data sets). There may be little value in accurately measuring parameters closely aligned with sloppy directions, and effort may be better spent measuring parameters aligned with stiff directions.

While the identification of both stiff and sloppy directions is desirable, the methods in this thesis are most capable of identifying the stiff directions. For this reason, these methods may be most useful to modellers wishing to conduct parameter space exploration, experimental design, or the discovery of important parameters. However, even if the sloppiest eigenvectors cannot be precisely computed, parameter reduction may be possible by moving in a sloppy "subspace", which can be derived from the two or three sloppiest eigenvectors (the subspace being the linear combinations of these eigenvectors). This is discussed further in Section 0.5.3.

0.3 Sloppiness vs Parameter Irrelevance

Many physical systems can be simplified by removing parameters or variables from equations. For example, despite air being compressible (as can be seen when pumping up a tyre), this phenomenon has very little effect on the aerodynamics of a car or aeroplane if its speed is well below the speed of sound. A variable representing the change in air density ($\partial\rho/\partial t$) can be set to zero, creating a simplified predictive model with virtually no loss in accuracy. Similarly, in the case of heavy weights falling short distances, drag coefficients (c_d) can be set to be zero. In both of these cases, the systems have been simplified by setting a variable or coefficient to zero, treating it as if it were irrelevant.

A model which exhibits sloppiness has an intrinsic form of "parameter redundancy", but this does not necessarily mean there are parameters that can simply be set to zero and removed from the model's equations. It may be the case that a model

is sensitive to changes in any individual parameter, but if a combination of parameters are jointly varied in a particular manner, the model's output may be virtually unchanged.

0.3.1 Illustrative Example

To illustrate the distinction between sloppiness and parameter irrelevance, consider a model represented by a scalar function f. This function depends on a parameter vector $\theta = [\theta_0, \theta_1, \theta_2]$, where

$$f(\theta) = \sin(\theta_0\theta_1) + \theta_2. \tag{0.5}$$

Suppose our starting point in parameter space is $\theta = [0, 1, 1]$. In this case, changes in θ_1 will be inconsequential, but changes in either of the other parameters will affect the model output. We can conclude that θ_1 is irrelevant, and the following vector is a sloppy direction of movement in parameter space, as it does not affect the model's output:

$$v_{\text{sloppy}} = [0, 1, 0]. \tag{0.6}$$

Conversely, a stiff direction in parameter space exists, which does significantly affect the value of f. This stiff direction is given by:

$$v_{\text{stiff}} = [0, 0, 1]. \tag{0.7}$$

In this (rather trivial) case, the sloppy and stiff directions align with the "bare" parameters (i.e., v_{sloppy} and v_{stiff} contain only one non-zero element each).

Now consider a different position in parameter space: $\theta' = [\sqrt{\pi}, \sqrt{\pi}, 1]$. A movement in the direction of v_{sloppy} will now change the output of f, so v_{sloppy} is thus not a perfectly sloppy direction of f at point θ'. However, we now have two new directions which are sloppy[1] for infinitesimal movements in parameter space:

$$u_{\text{sloppy},1} = [1, 0, -1] \tag{0.8}$$

and

$$u_{\text{sloppy},2} = [1, -1, 0]. \tag{0.9}$$

Here we have made use of the Taylor series approximation $\sin(x) \approx -x$, for $x \approx \pi$. The sloppy directions in this region of parameter space do not align with bare parameters. Thus, no one parameter is irrelevant (which would enable its value to be changed freely), but there are still directions of parameter insensitivity (sloppiness).

[1]Note that these are not necessarily the sloppiest directions in existence. However, they are sufficiently sloppy for illustrative purposes.

0.4 Deriving Model Sloppiness

To introduce the concept of model sloppiness, we begin by considering least-squares fitting – a foundational statistical technique, which generates a cost function to minimise when fitting a model to data. We then take this cost function, describe how it can be linked to other statistical concepts – including the Fisher Information Matrix – and show how we can derive sloppy and stiff directions in parameter space by carrying out an eigendecomposition. In Section 0.5, we present an alternative perspective, which allows us to interpret model sloppiness in a parameter-independent way.

0.4.1 Least-Squares Fitting

In statistics and machine learning, least-squares fitting is a straightforward method of calibrating models to data. If we have empirical observations (x_i, y_i), and wish to fit a parameterised model $f(\theta, x)$ to that data, we may estimate θ by finding the value which minimises a quadratic cost function of the form:

$$\sum_i (y_i - f(\theta, x_i))^2. \tag{0.10}$$

The value which minimises the cost function is the least-squares estimate for θ.

Let us define a nondeterministic data generating process, $y(\theta, t)$, which produces an N-channel multivariate time series when given a parameter vector θ. Each channel could represent a different quantity of interest – for example, the channels could be GDP, inflation, and unemployment. Channels could also represent different trials of the same experiment (e.g., three trials of an experiment with three observable variables could be represented as a time series with $N = 9$ channels). Additionally, let $f(\theta, t)$ be a deterministic function which takes the same parameters as y. The functions y and f have the form:

$$f(\theta, t) = [f_1(\theta, t), f_2(\theta, t), \ldots, f_N(\theta, t)] \tag{0.11}$$

and

$$y(\theta, t) = [y_1(\theta, t), y_2(\theta, t), \ldots, y_N(\theta, t)] \tag{0.12}$$

i.e., they are N-channel time series functions, where $\theta \in \mathbb{R}^M$ is an M-element parameter vector of real numbers, and $t \in \{1, 2, \ldots, T\}$ represents the time step index. The function f is deterministic, while y is defined as f with additive Gaussian noise:

$$y_\alpha(\theta, t) = f_\alpha(\theta, t) + \varepsilon_{\alpha,t}, \tag{0.13}$$

for all $\alpha \in \{1, 2, \ldots, N\}$ and $t \in \{1, 2, \ldots, T\}$. The values $\varepsilon_{\alpha,t}$ represent Gaussian noise components for channel α at time step t, and are assumed to be mutually independent.

Suppose we have empirical observations, $d_{\alpha,t}$, and we wish to fit the model (f) to these observations. Under the assumption that $\varepsilon_{\alpha,t}$ are normally distributed (with variances $\sigma^2_{\alpha,t}$), computing the maximum likelihood estimator (MLE) for θ corresponds to minimising a quadratic cost function, which we will call χ^2:

$$\chi^2(\theta) = \frac{1}{2NT} \sum_{\alpha=1}^{N} \sum_{t=1}^{T} \left(\frac{f_\alpha(\theta, t) - d_{\alpha,t}}{\sigma_{\alpha,t}} \right)^2. \tag{0.14}$$

Note that because $\varepsilon_{\alpha,t}$ are normally distributed, this cost function is equal to the log-likelihood minus a constant term.

Instead of considering the quadratic cost function associated with fitting a model to empirical data, now consider the quadratic cost associated with the model being run with two different parameter vectors, θ and θ^*:

$$\chi^2(\theta, \theta^*) = \frac{1}{2NT} \sum_{\alpha=1}^{N} \sum_{t=1}^{T} \left(\frac{f_\alpha(\theta, t) - f_\alpha(\theta^*, t)}{\sigma_{\alpha,t}} \right)^2 \tag{0.15}$$

Now consider the Hessian of χ^2:

$$H_{ij}(\theta) = \frac{\partial^2 \chi^2}{\partial \theta_i \partial \theta_j} \tag{0.16}$$

$$= \frac{1}{2NT} \sum_{\alpha=1}^{N} \sum_{t=1}^{T} \frac{\partial^2}{\partial \theta_i \partial \theta_j} \left(\frac{f_\alpha(\theta, t) - f_\alpha(\theta^*, t)}{\sigma_{\alpha,t}} \right)^2 \tag{0.17}$$

$$= \frac{1}{NT} \sum_{\alpha=1}^{N} \sum_{t=1}^{T} \frac{1}{\sigma^2_{\alpha,t}} \frac{\partial f_\alpha(\theta, t)}{\partial \theta_i} \frac{\partial f_\alpha(\theta, t)}{\partial \theta_j} + \frac{f_\alpha(\theta, t) - f_\alpha(\theta^*, t)}{\sigma^2_{\alpha,t}} \frac{\partial^2 f_\alpha(\theta, t)}{\partial \theta_i \partial \theta_j}. \tag{0.18}$$

For $\theta \approx \theta^*$, the second derivative term vanishes and we are left with

$$H_{ij}(\theta) \approx \frac{1}{NT} \sum_{\alpha=1}^{N} \sum_{t=1}^{T} \frac{1}{\sigma^2_{\alpha,t}} \frac{\partial f_\alpha(\theta, t)}{\partial \theta_i} \frac{\partial f_\alpha(\theta, t)}{\partial \theta_j}. \tag{0.19}$$

When making this approximation, we assume that f_α is smooth, and well-approximated by it's first-order Taylor expansion. Additionally, we assume θ is close enough to θ^* that the second derivative is small, when weighted by weighted by $f_\alpha(\theta, t) - f_\alpha(\theta^*, t)$. When these conditions do not hold, the approximated Hessian will be inaccurate.

As we will discuss in the next section, the Hessian is closely linked to deriving the stiff and sloppy eigenvectors, via the Fisher information matrix. If the Hessian

matrix is inaccurate, the eigenvectors we derive from it will not closely align with the true stiff and sloppy directions. Because the eigenvectors are inaccurate – being comprised of some linear combination of the "true" eigenvectors of the model – the vectors estimated to be stiffest/sloppiest will not cause the largest/smallest change in model outputs. As a result, the derived eigenspectrum will be compressed, in the sense that it will not span the full range the true eigenvalues.

0.4.2 Information Geometry

Information geometry is a field which links probability theory with differential geometry [114]. A set of probability distributions parameterised with different parameter values are considered to be sets which form a manifold in a geometric space, which provides a way to define "distances" between distributions [114, 3]. This allows problems in statistics to be framed in a way that provides alternative perspectives and insights into areas including statistical inference, optimisation, and machine learning.

To analyse the model f from the perspective of information geometry, let us begin by defining the function's individual scalar outputs by a single index (k), rather than two (α, t). Let us do this via the following definitions:

- ξ_A is the set of all possible α values.
- ξ_T is the set of all possible t values.
- ξ_K is the Cartesian product of ξ_A and ξ_T i.e. $\xi_K = \{(\alpha, t) : \alpha \in \xi_A, t \in \xi_T\}$.

We now have a way to index the model outputs $f_\alpha(\theta, t)$ by a single scalar index, $f_k(\theta)$, where $k \in \xi_K$.

When the errors $\varepsilon_{\alpha,t}$ (now written ε_k) are normally distributed, we can consider the data generating process of equation 0.13 as a multivariate normal distribution:

$$X(\theta) \sim \mathcal{N}(\mu(\theta), \Sigma), \tag{0.20}$$

where $X(\theta)$ is a k-element vector, representing a realisation of the function $y_k(\theta) = f_k(\theta) + \varepsilon_k$. Let μ_k be the expected value of the kth element of X, which is:

$$\mu_k(\theta) = \mathrm{E}[y_k(\theta)]. \tag{0.21}$$

Since $\mathrm{E}[\varepsilon_k] = 0$, we have:

$$\mu_k(\theta) = f_k(\theta). \tag{0.22}$$

The covariance matrix Σ is diagonal, as per the assumption that ε_k are independent.

In the sections below, we define the Fisher information matrix (FIM) and KL divergence, which play important roles in the computation of stiff and sloppy directions [121]. While the FIM can be seen as solely a useful tool we calculate as an intermediate step when deriving stiff and sloppy directions, it can also provide a deeper geometric perspective on model sloppiness. This interpretation, which allows us to gain greater insight into the behaviour of a model, is discussed in Section 0.5.2.

Fisher Information Matrix

The FIM is defined as [140]:

$$I_{ij}(\theta) = \mathrm{E}\left[\left(\frac{\partial}{\partial\theta_i}\log p(X;\theta)\right)\left(\frac{\partial}{\partial\theta_j}\log p(X;\theta)\right)\,\middle|\,\theta\right]. \tag{0.23}$$

For a multivariate normal distribution, the FIM is given by:[2]

$$I_{ij}(\theta) = \left(\frac{\partial\mu(\theta)}{\partial\theta_i}\right)^T \Sigma^{-1}\left(\frac{\partial\mu(\theta)}{\partial\theta_j}\right), \tag{0.24}$$

where the mean vector and covariance matrix are:

$$\mu(\theta) = \begin{bmatrix}\mu_1(\theta)\\ \mu_2(\theta)\\ \vdots\\ \mu_K(\theta)\end{bmatrix} \tag{0.25}$$

$$\Sigma = \begin{bmatrix}\sigma_1^2 & 0 & \cdots & 0\\ 0 & \sigma_2^2 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & \sigma_K^2\end{bmatrix}. \tag{0.26}$$

We can then represent the FIM as:

$$I_{ij}(\theta) = \sum_{k=1}^{K}\frac{1}{\sigma_k^2}\frac{\partial\mu_k(\theta)}{\partial\theta_i}\frac{\partial\mu_k(\theta)}{\partial\theta_j}, \tag{0.27}$$

which is a constant multiple of equation 0.19. Thus, the cost Hessian is approximately equal to the FIM (multiplied by a constant), under the assumption that ε_k are independent and normally distributed:

$$I_{ij}(\theta) \approx H_{ij}(\theta) \times \text{constant}. \tag{0.28}$$

[2]See Appendix A.1 for proof.

Kullback–Leibler Divergence and the FIM

We now introduce the Kullback-Leibler (KL) divergence. In discrete form, the KL divergence between two distributions P and Q is defined as follows [92, 58]:

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \ \log\left(\frac{P(x)}{Q(x)}\right). \tag{0.29}$$

The KL divergence is a measure of the "distance" between two distributions (see A.2 for definitions and further discussion of KL divergence). The nondeterministic models we consider in this thesis are functions that produce stochastic outputs, and can be thought of as multivariate distributions (with $K = |\xi_K|$ variables) that change when θ is varied. Hence, the KL divergence is a useful measure of the change in model outputs that results from altering θ.

Now consider P to be a distribution parameterised by θ (call this P_θ). The FIM is related to the KL divergence by its second derivative [52]:

$$I_{ij}(\theta) = \left.\frac{\partial^2 D_{\mathrm{KL}}(P_{\theta+\delta\theta} \parallel P_\theta)}{\partial\theta_i \partial\theta_j}\right|_{\delta\theta=0} \tag{0.30}$$

i.e., the FIM is a local measure of the curvature of the KL divergence surface in parameter space. This characteristic gives us a second-order approximation for the divergence induced by perturbing θ by a small vector $\delta\theta$:

$$D_{KL}(P_{\theta+\delta\theta} \parallel P_\theta) \approx \frac{1}{2}\delta\theta^T I(\theta)\delta\theta, \tag{0.31}$$

where P_θ is a distribution (model) parameterised by θ.

If we wish to find the directions that change the model output distribution the most, we compute its eigenvectors and eigenvalues using principal component analysis (PCA). The FIM's PCA decomposition yields a hierarchy of sensitive/insensitive directions (eigenvectors) in parameter space [121] – i.e., the stiff/sloppy directions we wish to find. The magnitude of the eigenvalues indicates the relative stiffness of each respective eigenvector, with larger eigenvalues corresponding to stiffer eigenvectors.

Suppose we compute the normalised eigenvectors v_m, and their corresponding eigenvalues λ_m. Consider perturbing θ in the direction $\delta\theta = \eta v_m$, where η is a small positive scalar:

$$D_{KL}(P_{\theta+\delta\theta} \parallel P_\theta) \approx \frac{1}{2}(\eta v_m)^T I(\theta)\eta v_m \tag{0.32}$$

$$= \frac{1}{2} v_m^T \lambda_m v_m \eta^2 \tag{0.33}$$

$$= \frac{1}{2}\lambda_m \eta^2 \tag{0.34}$$

i.e., when θ is perturbed in the direction of v_m, the induced KL divergence is proportional to the corresponding eigenvalue λ_m.

0.4.3 Non-Gaussian Cases

For a nondeterministic model whose stochastic component is not Gaussian (as is often the case in agent-based modelling [64]), the cost Hessian (H) is not equal to the FIM. Additionally, the values of the variances (σ_k^2) may be unknown, requiring heuristic values to be used in their place (discussed further in Section 0.4.5). Despite these issues, researchers have found H to be a workable approximation for the FIM, in the sense that it predicts the parameter sensitivities [112] even though it does not fully capture the non-Gaussian variability. The techniques shown to be valid in the Gaussian case also work successfully in a wide range of non-Gaussian cases [110, 64], and accordingly, we employ the cost Hessian throughout this thesis. However, as discussed in Section 0.4.1, if H is a poor approximation for the FIM, estimates of stiff and sloppy eigenvectors may be inaccurate, and eigenvalue spectra will be artificially compressed.

0.4.4 Sloppiness in Deterministic Models

So far, model sloppiness has been introduced as an inherently statistical concept, and the method used to compute stiff/sloppy directions has been justified in terms of statistical theory. However, the method has also been applied to deterministic models, in analysis of the directions in parameter space that lead to the greatest/least variation in the model's (deterministic) outputs [110]. This can be justified by considering χ^2 to be simply a cost function that quantifies the difference between two curves, and measuring the sensitivity of the cost function to changes in parameters. Alternatively, one can pretend we are dealing with a function equal to the deterministic function of interest combined with some imaginary Gaussian noise. As before, the cost function takes quadratic form:

$$\chi^2(\theta, \theta^*) = \frac{1}{2K} \sum_{k=1}^{K} \left(\frac{f_k(\theta) - f_k(\theta^*)}{\sigma_k} \right)^2, \tag{0.35}$$

where θ and θ^* are two (close) points in parameter space, and $f_k(\theta)$ is the kth output of a deterministic vector-valued function $f(\theta)$. The value of the scaling constant σ_k has no statistical meaning. Instead, σ_k is set according to a simple heuristic (such as

one of those described in Section 0.4.5 below) to make each channel of the time series of approximately equal order of magnitude.

Later in this thesis, we shall demonstrate the methods described above on deterministic functions for two purposes. In Chapter 1, we use the methods to estimate the sloppiness of a stochastic DSGE via computing the sloppiness of its IRF (which are deterministic, but intrinsically relate to the behaviour of a DSGE subject to stochastic shocks). In Chapter 2, we apply sloppiness analysis to deterministic neural network surrogates to investigate the sloppiness of the nondeterministic models they emulate.

0.4.5 Computing Sloppiness

In practice, computing the second derivative of H can be computationally intensive. For this reason, we follow other authors ([27, 112]) and use the approximation given by equation 0.19, i.e.,

$$H_{ij}(\theta) \approx \frac{1}{K} \sum_{k=1}^{K} \frac{1}{\sigma_k^2} \frac{\partial f_k(\theta)}{\partial \theta_i} \frac{\partial f_k(\theta)}{\partial \theta_j}. \tag{0.36}$$

That is, H_{ij} is approximated by the product of Jacobian matrices:[3]

$$H(\theta) \approx \frac{1}{K} J(\theta)^T J(\theta) \tag{0.37}$$

where

$$J_{ki}(\theta) = \frac{\partial}{\partial \theta_i} \left(\frac{f_k(\theta)}{\sigma_k} \right). \tag{0.38}$$

Output Normalisation

When computing the stiff/sloppy directions of (a) models with no stochastic component, (b) models possessing stochasticity that is not additive Gaussian noise, or (c) models without known structure to their stochasticity, scaling constants can be used for $\sigma_{\alpha,t}$ in place of standard deviations. The values for these constants are often chosen such that the values $f_\alpha(\theta, t)/\sigma_{\alpha,t}$ are of similar magnitude. Two common choices for $\sigma_{\alpha,t}$ are the maximum possible value f_α can take [68], and the average value of f_α over many simulated runs [112]. It is not uncommon to use a single normalisation constant for all time periods of a given channel, i.e. $\sigma_{\alpha,t} = \sigma_{\alpha,t'}, \forall t, t' \in \{1, 2, ..., T\}$. This approach can be justified by the desire to ensure the individually meaningful time series (i.e., individual channels such as GDP, inflation, unemployment) are of

[3] By convention, the Jacobian product is written $J^T J$, rather than JJ^T. This is common within literature on model sloppiness.

similar importance when computing sloppiness, but large proportional variations in the smallest values of each time series channel do not dominate the computation of sloppiness. For example, the difference between 3% and 4.5% GDP growth may be more significant than the difference between 0.5% and 0.75%, even though they represent the same proportional change.

The ad hoc nature of these heuristics does not appear to inhibit the computation of stiff/sloppy directions, despite a lack of statistical justification. For the purpose of this thesis, max-value normalisation ($\sigma_{\alpha,t} = \max_{t'}(f_\alpha(\theta^*, t'))$) will be used unless otherwise stated.

Parameter Normalisation

Because a model's parameters have different units and orders of magnitude, it is common to reparameterise the model of interest to use the logarithm of its original parameters [68, 112]. Doing so allows eigenvectors to be interpreted as proportional changes in parameters, and can help avoid issues of numerical instability [27]. This technique is used here in Chapter 1.

An alternative technique, which we employ in Chapters 2 and 3, is to choose upper and lower bounds for each θ_i, and reparameterise the model linearly: let $\hat{\theta}_i = a\theta_i + b$, where a and b are chosen such that $\hat{\theta} \in [0, 1]$ covers the desired range of possible θ_i values. This technique is further discussed in Section 2.4.1. This reparameterisation is especially useful when working with neural networks, where it is desirable for all inputs to be of similar magnitude and lie in a similar range [77].[4]

0.5 Behaviour Space, and the Meaning of Parameter Sloppiness

In this section, we go beyond describing how stiff and sloppy directions are computed, and instead investigate how sloppiness can be interpreted. Below, we discuss the concept of "behaviour space", how it differs from parameter space, model "manifolds" which sit within this space, and how sloppiness presents itself when we consider it from this perspective. In this section, we will assume the function (model) of interest is deterministic, i.e., $y(\theta, t) = f(\theta, t)$.

[4]Normalising inputs has been shown to lead to faster convergence during training. While many normalisation techniques involve scaling inputs so they approximately follow a standard normal distribution, we use $[0, 1]$ normalisation, which has the advantage of placing strict bounds on the values the inputs can take when we are building our training data sets.

0.5.1 Behaviour Space and Model Manifolds

Let us define parameter space, Θ, as a vector space which contains all allowable parameter values a model can accept. If a modeller can choose the allowable parameter values for their model, then in principle, the parameter space may be freely chosen, as long as the model f is defined for every vector $\theta \in \Theta$. Let us also define behaviour space (also known as "data space", or "output space"), as a vector space which contains the outputs from f. If the function produces vector outputs in the set $\mathbb{R}^K$ (or some subset of it), then behaviour space could be defined as $\mathbb{R}^K$. However, it is not necessarily true that all points in behaviour space are plausible outputs of the model f. For example, if the model is represented by the one-dimensional function $f(x) = x^2$, $x \in \mathbb{R}$ not all points in $\mathbb{R}$ are possible outputs. (Specifically, the possible outputs of f lie in $[0, \infty)$). The set of possible outputs that f may generate is sometimes referred to as the "model manifold" [121], which is a non-strict subset of behaviour space.

To investigate the concept of behaviour space and model manifolds, consider the following function, which takes a two-element parameter vector, and outputs a two-element vector:

$$f(\theta, t) = e^{-\theta_1 t} + e^{-\theta_2 t}, \tag{0.39}$$

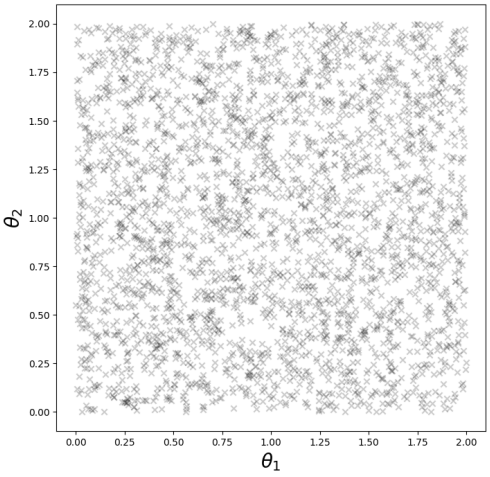
where $\forall t \in \{1, 2\}$, and $\theta_1, \theta_2 \in [0, \infty)$. This is a two-dimensional version of a function commonly used to illustrate the concept of model manifolds [150, 149]. Figure 1 illustrates the difference between parameter space and behavioural space for this function.

Before this section, we have considered how to derive sloppy and stiff directions in parameter space, and interpreted them indicating how a model is affected by variations in its parameters. We can, however, consider sloppiness in behaviour space as well. Many models can be reparameterised in different ways – for example, by changing coordinates, or scaling parameters – and often it is arguable that there are multiple parameterisations which are as valid as one-another[5] [121]. If sloppiness is derived from derivatives with respect to the chosen parameters, it seems possible, a priori, that any sloppiness observed may only be a result of the parameterisation choices [106]. However, considering behaviour space provides a way to analyse sloppiness in a parameterisation-independent way, as the model manifold is independent of the chosen parameterisation [121].

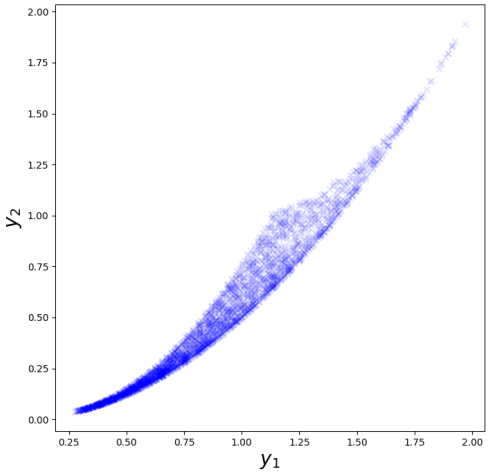
[5]For example, models of radioactive decay can be parameterised using the decay rate, or the expected survival period [121]. These are both physically meaningful, and it is difficult to justify one being more fundamental than the other.

It has been widely observed that the model manifolds of sloppy functions have distinctive features when plotted in behaviour space. The most prominent of these features a "ribbon-like" structure, where some dimensions of the manifold are much wider than others [150]. For a model with many outputs – which therefore has a high-dimensional behaviour space – the model manifold is often observed to resemble a multi-dimensional "hyper-ribbon" structure. The hyper-ribbons are named as such because they possess a hierarchy of dimensions, making them analogous to a three-dimensional ribbon [149] (which we think of as being much longer than it is wide, and much wider than it is thick). When sloppy and stiff directions are computed and mapped onto behaviour space (and the model manifold), they often appear aligned with the axes of the hyper-ribbon [149] (as observed in Figures 1c and 1d). Additionally, the ratios of the widths of the hyper-ribbon dimensions has been found to correspond closely to the ratios of the square roots of the eigenvalues [148], providing a geometric interpretation of eigenvalue spectra.

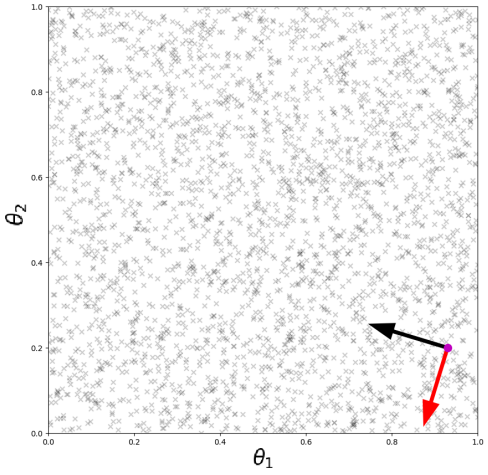
Figure 1 shows the model manifold for the exponential function (equation 0.5.1), which has a ribbon-like form, with a "length" dimension (bottom-left to top-right) significantly larger than the "width" dimension (top-left to bottom-right). When viewed in behaviour space, the sloppy eigenvector (computed via the eigendecomposition of $J^T J$) is directed across the smallest dimension of the ribbon, while the stiff directions is directed along the longer dimension. Despite the eigenvectors being of equal length in parameter space (i.e. perturbations are equal in magnitude, in parameter space), they are of very different lengths when mapped onto behaviour space. This reflects the behavioural change being much greater for stiff perturbations than for sloppy ones, despite movements in parameter space being equal.



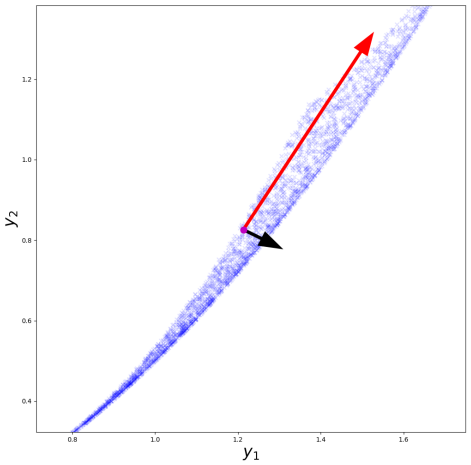
(a) Parameter Space



(b) Behaviour Space



(c) Parameter Space with Eigenvectors



(d) Behaviour Space with Eigenvectors

Figure 1: **Parameter space and behaviour space**, for exponential function $f(\theta, t) = \exp(-\theta_1 t) + \exp(-\theta_2 t), \forall t \in \{1, 2\}$. (a) Parameter space, with 2,000 points sampled uniformly from $[0, 2]^2$. (b) The corresponding 2,000 points in behaviour space ($y_1 = f(\theta, 1), y_2 = f(\theta, 2)$), showing a ribbon-like manifold. Although y_1 and y_2 can each take any value in $[0, 2]$ individually, not all points in two-dimensional space $[0, 2]^2$ are feasible outputs for f. Thus, the model manifold is a strict subset of $[0, 2]^2$ (c) Magnified parameter space, with equal-length eigenvectors shown for point $\theta^* = (0.93, 0.2)$. (d) Magnified behaviour space with the same stiff (red) and sloppy (black) eigenvectors, computed at the same point θ^*, mapped onto behaviour space. Equal-magnitude perturbations in parameter space (along the eigenvectors) can cause changes of very different magnitudes in behaviour space, reflecting the relative stiffness of the eigenvectors.

0.5.2 Moving Within the Model Manifold

It has been widely observed that the boundaries of a model's manifold in behaviour space correspond to simpler, alternative models [151], in the sense that this alternative model requires fewer parameters than the original. These simplifications can correspond to eliminating a parameter completely (e.g., by setting θ_1 or θ_2 to zero), or "folding" two parameters into one (setting $\theta_1 = \theta_2$, and only allowing θ_1 to vary [121]). These boundaries not only provide us with more compact models – they can also give us insight into the nature of the model of interest. For example, in physical models that incorporate many mechanisms of action, following sloppy directions has led to the formation of simpler models representing well-understood approximations with known regimes of validity [148] (such as the zero-compressibility limit in low-velocity fluid dynamics).

Simplifying models (by following sloppy directions) and exploring different possible behaviours of a model (following stiff directions) both correspond to movements through parameter space, which can be mapped into movements along the model manifold [151]. As seen in the case of the exponential function above, model manifolds can have curved, nonlinear forms, with eigenvectors have directions across and along the manifold (which is especially common for curved hyper-ribbon manifolds). The eigendirections can change as one moves towards the manifold boundaries (or along the longest dimension of the manifold). This illustrates the need for recomputing Jacobian many times: the eigenvectors of $J^T J$ (and the directions towards manifold boundaries) will change after each step along stiff or sloppy directions. This is the core justification for the motivating problem of this thesis: the need to quickly compute (and recompute) stiff/sloppy eigenvectors as we move through parameter (and behaviour) spaces.

Interpreting $J^T J$

Previously, we considered $J^T J$ to be an approximation for the FIM, which justified using it to compute sloppy and stiff eigendirections from a statistical point of view. However, behaviour space and the model manifold provide an alternative perspective on $J^T J$. To derive this, let us consider the Euclidean distance, ds, between two infinitesimally close points in the (multi-dimensional) behaviour space, y and $y + dy$.

$$ds^2 = \sum_k (dy_k)^2 \tag{0.40}$$

$$= dy \cdot dy. \tag{0.41}$$

As the Jacobian matrix is defined as $J_{ki} = dy_k/d\theta_i$, we have:

$$dy = J\, d\theta. \tag{0.42}$$

Thus, we can write:

$$ds^2 = (J\, d\theta) \cdot (J\, d\theta) \tag{0.43}$$

$$= d\theta^T J^T J\, d\theta. \tag{0.44}$$

Formally, this makes $J^T J$ an induced metric on the model manifold [150, 148, 121], measuring distances on the manifold based on movements in parameter space. An alternative, informal interpretation is that $J^T J$ acts as a matrix which transforms (via rotations and distortions) regions of parameter space into behaviour space. The distortion process represented by the matrix causes eigenvectors derived in parameter space (using one of many parameterisations) to transform into vectors in behaviour space, with changed magnitudes and orientations. Differently-parameterised models will produce different values for the $J^T J$ matrix, but these matrices will all appropriately distort points in their respective parameter spaces into the same model manifold, with the sloppy and stiff directions in behaviour space maintained [121]. While $J^T J$ is parameterisation-dependent, it will have a similar structure for all parameterisations, including very similar eigenvalue spectra [150], reflecting the intrinsic geometry of the model manifold (including the hyper-ribbon structure, observed in sloppy models).

0.5.3 Sloppy Subspaces

If two (or more) FIM eigenvalues are extremely small, and of similar magnitude, then any linear combination of their respective eigenvectors will be very sloppy. Because of noise present when computing the FIM (including that resulting from stochasticity in ABMs, causing variation each time the model is run), determining the ordering of the eigenvalues may difficult, and they may sometimes may be "swapped" due to noise. If the eigenvalues are degenerate (or approximately degenerate), any linear combination of their respective eigenvectors (v_i, v_j) will be as sloppy as the eigenvectors themselves. This results in the sloppy eigenvectors being less meaningful individually, as their alignment with individual parameters – which would usually indicate parameter unimportance – will be arbitrary (and potentially most influenced by statistical noise, or small inaccuracies in numerical methods). For this reason, it can be more helpful to consider the sloppy "subspace" [148] – formed by all linear combinations of the equally-sloppy (or approximately equally-sloppy) eigenvectors – as

being a more useful way to infer properties of the model, including parameter importance. If one can determine the subspace, it is possible to attempt model reduction and phase exploration, even if eigenvectors are generally not accurately identifiable. Model reduction may be possible by finding vectors in this subspace that lead towards model manifold boundaries (often corresponding to reduced models [151]), and vectors orthogonal to this subspace may be used to move along the model manifold in what are effectively stiff directions.

0.6 Challenges Resulting from Sloppiness

The existence of sloppiness in a many-parameter model can be a mixed blessing for the modeller. Sloppiness can make parameter identification difficult or impossible [148], especially for parameters that individually align closely with the sloppiest eigenvectors. Thus, a researcher wishing to determine precise values for all parameters in a model may face an extremely difficult task, requiring an immense volume of data. However, sloppiness can indicate a model may be simplified to reduce the parameter count with little or no effect on the model's predictions. MBAM is capable of doing this in a semi-automated way [151]. Additionally, identifying stiff directions can assist with the exploration of parameter space to find distinct transitions in the behaviour of the model of interest [112].

0.6.1 Other Methods for Reducing Parameter Space

Challenges related to model sloppiness – and the desire to reduce parameter count – have been approached in different ways, including methods where the problem at hand is framed through the lens of model sloppiness theory. In addition to MBAM and other manifold-based techniques, different methods have been developed to reduce the dimensionality of models. Below, we describe two of the simplest and most commonly used methods.

Regularisation and Model Selection Techniques

These techniques are commonly applied in the context of building linear models, although they are more generally applicable. Consider designing a model which is to be fit to data (data which may be empirical, or generated from a larger model). It is common in economics and other fields to begin with a linear regression model of the form:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + ... + \beta_n x_n + \varepsilon, \tag{0.45}$$

where x_i are variables representing potentially relevant quantities, and β_i are the model's parameters. By including a large number of variables, the initial parameter count is high. However, only some variables will have a significant effect on the model's outputs. As a result, some parameters can be set to zero, with little impact on the model's outputs. Notable methods of doing so include lasso regression [147], where the sum of the parameter values' magnitudes is constrained, to encourage some of the β_i values to go to zero, while maintaining good fit. Alternatively, statistical model selection criteria (such as the Akaike information criterion, AIC) can be used to balance goodness-of-fit against model complexity, allowing parameters to be sequentially eliminated until the information criterion is minimised [143].

Principal Component Analysis

High-dimensional data often contains correlations between dimensions. It is possible that knowledge of the location of a data point in several dimensions is sufficient to accurately predict its position in the remaining directions. When such relationships are linear, PCA is able to construct low-dimensional representations from an eigendecomposition, with the highest-eigenvalue eigenvectors representing directions of maximum variance in the data [79]. While PCA is a linear dimension-reduction process, nonlinear forms also exist, including the use of neural networks [90, 40], and Kernel PCA. The latter involves first transforming the data nonlinearly, in order to make directions of greatest variation approximately linear in this new space [109], after which conventional PCA can be applied.

0.7 Motivating Problems for this Thesis

Computing the stiff/sloppy directions of a nondeterministic model requires estimating Jacobians using the stochastic finite difference (SFD) method: the model is run many times at the parameter point of interest, θ, then many times at perturbed points in parameter space ($\theta + \delta\theta$) to compute the derivatives of the model with respect to each parameter. This method may require numerous simulations at the initial and perturbed points in parameter space, but such is usually achievable.

However, computing a model's sloppy/stiff eigenvectors at many different points in parameter space remains desirable in many cases (e.g., model simplification, model fitting, and parameter space exploration). To accomplish this, many simulations will need to be carried out at each point of interest in parameter space. The number of simulations required is not known in advance, and the locations (in parameter

space) of the required simulations cannot be known in advance (as the ith location in parameter space depends on the result of computing the sloppy eigenvectors at the $(i-1)$th point). If model simplification is to be attempted starting from a different point in parameter space, the whole process of stochastic finite differencing will need to be repeated, potentially taking just as long as before. The workflow can be described as follows:

Method 0 (Conventional)

1. Task 1: Simplify model starting at θ.
    - **(Slow)** Perform many simulations at θ, then at θ', then θ'', ...
2. Task 2: Simplify model starting at ϕ (a completely different location in parameter space).
    - **(Slow)** Perform many simulations at ϕ, then at ϕ', then ϕ'', ...
3. Task 3: Simplify model starting at another new point, ψ.
    - **(Slow)** Perform many simulations at ψ, then at ψ', then ψ'', ...

For each new model simplification task to be run, the full process of repeatedly carrying out stochastic finite differencing must be repeated.

These are the methods outlined in the subsequent parts of this thesis:

Method 1 (IRFs)

1. Task 1: Simplify the model starting at θ.
    - **(Moderate)** Compute the IRFs at θ and θ^+, in order to numerically differentiate the (deterministically computed) IRF curves. θ, then at θ', then θ'', ...
2. Task 2: Simplify model starting at ϕ.
    - **(Moderate)** Same as above, but at ϕ, then at ϕ', then ϕ'', ...
3. Task 3: Simplify model starting at ψ.
    - **(Moderate)** Same as above, but at ψ, then at ψ', then ψ'', ...

Method 2 (Surrogates)

0. Task "0": Train a neural network surrogate with many training examples. This network should be capable of estimating the model output (or its FIM) at arbitrary points in parameter space.
    - **(Slow)** The model is run many times to create training examples, the network is trained (a process that may be time intensive)
    - However, the training process needs to be carried out only once.
1. Task 1: Simplify the model starting at θ.
    - **(Fast)** Use the trained network to numerically (or analytically) compute the derivatives at θ, then at θ', then θ'', ...
2. Task 2: simplify model starting at ϕ.
    - **(Fast)** Same as above, but at ϕ, then at ϕ', then ϕ'', ...
3. Task 3: simplify model starting at ψ.
    - **(Fast)** Same as above, but at ψ, then at ψ', then ψ'', ...

While training the neural network (Task 0) may be slow, once it is trained, movement through parameter space can be carried out quickly (assuming a differentiable neural network architecture is chosen). Note that the training examples should provide good coverage of the relevant parameter space, as discussed further in Section 2.8.4.

The above methods for the computation of sloppy directions to conduct model simplification involve processes that can be applied in parameter space exploration, simply using the stiff eigendirections, rather than the sloppy, for movements in parameter space.

0.7.1 Exploration of Parameter Space

As previously discussed, another application of sloppiness computation is the exploration of parameter space, often to find regions where the model behaves differently from its known patterns [112]. This exploration typically requires repeatedly computing the stiff eigenvectors, rather than sloppy eigenvectors. The methods described in this thesis are especially well-suited to this task, as stiff eigenvectors were generally

found to be easier to identify than sloppy ones.[6]

0.8 Conclusion

This chapter outlines the purpose of this thesis: to aid in the development and investigation of economic models via the investigation of model sloppiness. The models we focus on primarily are conventional DSGE models and ABMs, due to their positioning within economics. Developing methods applicable to DSGEs is worthwhile due to the widespread use of such models by mainstream economic institutions, particularly central banks. Simultaneously, ABMs are at the forefront of research into the economy from a complex systems perspective, and have demonstrated promising capabilities (including as direct competitors to DSGEs [118]), suggesting that new techniques to analyse them could be highly relevant to the field.

The next chapter of this thesis describes and evaluates a novel method to compute model sloppiness in DSGE models. Later parts of this thesis focus on the use of machine-learning surrogates to create methods applicable to a wider range of models.

[6]The relative ease of identifying stiff directions compared to sloppy ones is discussed in section 2.8.2

Chapter 1

Sloppiness in DSGE Models

A DSGE model is a parameterised mathematical system, able to simulate a hypothetical economy using a set of economically interpretable rules. These rules typically stem from modern economic theories, including "bottom-up" rational expectation-based behavioural rules (from New Classical economics) and price "stickiness" (closely linked to New Keynesian economics).

DSGEs are now a mainstay of modern computational economic modelling [21, 48]. DSGE research is a large and active subfield of economics, and such models are in widespread use globally throughout many central banks and economic institutions, including the European Central Bank (ECB), the US Federal Reserve, and the International Monetary Fund (IMF) [129].

1.1 Literature Review: DSGE Models

In this section, we review the key aspects of DSGE modelling. This will include the history of DSGEs, their structure, applications, and the criticisms surrounding their use.

1.1.1 History

For several decades after the Second World War, econometric models with empirically estimated parameters were dominant in macroeconomics. Among them were simultaneous equation models (SEMs) where economic relationships between variables could be encoded in parameterised equations. A simple example of an economic SEM model describing the market for a good is given by:

$$Q_d = \alpha - \beta P + \gamma Y \tag{1.1}$$

$$Q_s = \psi + \phi P - \zeta C_p \tag{1.2}$$

$$Q_d = Q_s. \tag{1.3}$$

Equation 1.1 specifies a relationship between the demand for a good (Q_d), its sale price per unit (P), and consumers' pre-tax income (Y). Equation 1.2 relates the total quantity of the good firms are willing to supply (Q_s) to the sale price (P) and cost of production per unit (C_p). Finally, equation 1.3 represents market equilibrium (demand and supply equal one another, assumed to occur when firms adjust their prices until this equality holds). If we enforce $\beta, \gamma, \phi, \zeta \geq 0$, then positive and negative relationships between variables can be captured (e.g., consumer demand depends negatively on price and positively on income). The parameters of SEMs could be estimated using empirical data, and used to predict the effect of policy changes (e.g., a new policy subsidising the production of a good, lowering C_p).

The economic consensus around using such models came under serious strain in the 1970s, partly caused by the difficulty of then-standard economic theory to explain contemporary economic crises (including, most notably, the period of "stagflation" in the United States, where meagre economic growth was accompanied by a high rate of inflation, something unusual in post-war USA). A major shift in economic thought developed, partly brought on by the publication of the "Lucas Critique" [102]. Lucas argued:

> *Given that the structure of an econometric model consists of optimal decision rules of economic agents, and that optimal decision rules vary systematically with changes in the structure of series relevant to the decision maker, it follows that any change in policy will systematically alter the structure of econometric models.*

That is to say, if a government uses a calibrated model to guide policy decisions (such as subsidising the production of a good, lowering C_p), the change in government behaviour can alter the parameters of the economic model. For example, if the subsidy is funded by government debt, and consumers anticipate that income tax rates will later rise to pay off this debt, the relationship between demand and income (equation 1.1) may change (specifically, parameter γ may fall, as consumers may be more wary of spending their income Y given that higher tax rates are expected in the future). Thus, the prediction the calibrated SEM makes regarding the policy's impact on production will be inaccurate, as the previously-estimated γ value is no longer valid. The value of γ cannot be re-estimated until after data is collected following the implementation of the policy.

This shift led to the widespread acceptance that macroeconomic models should be "micro-founded" [152, 78], i.e., the macroeconomic model's equations should be defined by rules specifying the behaviour of individual agents (people, firms, and governments). The stability of micro-founded models stems from their assumption that agents act rationally and possess rational expectations about the choices of other agents and policymakers. This feature ensures agents have made best-response decisions consistent with model predictions: the modeller is predicting agents' best responses with the same information that agents use to compute their best responses. An agent who observes the model's predictions becomes no better informed, so their actions do not change. In the case of our SEM, this would require us to model households' decision-making processes, namely those leading them to set their own γ value, which requires modelling the effect government announcements have on expectations of future income (Y).

While the perceived need for micro-foundations was not the only cause of the development of DSGEs, it was an important catalyst. Since then, such models have become dominant in mainstream macroeconomic modelling. However, the credibility of such models has come into serious doubt since the onset of the Global Financial Crisis of 2008.

Controversy Over DSGEs

DSGEs have become controversial, both in their ability to forecast the economy [36] and in their theoretical underpinnings [153]. Opposition has been raised against common features of DSGE models, including the assumption that unique equilibria exist, involving all markets in the economy reaching a stable "Steady State" in the absence of shocks. Additionally, the assumption of rational utility maximisation, which stipulates that agents act as rational optimisers (making decisions which solely maximise and expected utility function), has been criticised. Utility is usually [47, 127] assumed to be a function of an individual's consumption of goods and services, and leisure time. Socio-economic factors, such as economic inequality and economic positioning compared to one's peers, are often assumed to be irrelevant. Historically, models have often lacked a financial sector, contributing to heightened criticism of DSGEs during and after the Global Financial Crisis of 2008. (However, models incorporating a financial sector have since been developed, along with highly complex models addressing certain critiques of DSGEs [25].)

The analysis of the advantages and flaws of DSGEs is critical in economic forecasting and policy analysis. However, this thesis will not analyse the merits and

criticisms of DSGE models, and will instead focus on methods of analysing these models through the lens of model sloppiness. Because DSGE models are likely to remain in use by economic institutions worldwide, developing methods to analyse these models is worthwhile.

1.1.2 DSGE Structure/Design

DSGEs consist of a set of equations relating economically meaningful variables together. Common economic variables (with the symbols commonly used for each) include:

- Capital[1], K
- Consumption, C
- Economic output (GDP), Y
- Exports, X
- Imports, M
- Government spending, G
- Inflation, π
- Interest rate, r
- Investment, I
- Labour (number of workers), L
- Terms of trade, q
- Total factor productivity, Z
- Unemployment, U
- Wage rate, W

The equations governing the dynamics of a DSGE represent interactions between some combination of the economic entities described below.

[1]In economics, "capital" does not refer to money saved (which is instead known as "financial capital"). Instead, capital in economics generally refers to tangible assets such a machinery, buildings, and IT equipment, which are used to produce other goods and services [124]. A household may own capital directly, or via financial securities (e.g., owning shares in a company that leases agricultural equipment to farms).

Households

Households are representative of workers, who provide labour to firms, and consume goods and services. Households earn wages from the labour they provide to firms, and income from capital they own. They are also assumed to possess a utility function, which they seek to maximise through rational decision-making. Utility is usually a concave function of consumption and time spent on leisure, where leisure refers to activities other than labour [47, 127].

Firms

Firms use capital and labour as inputs, and produce goods then sold to the market. Prices are chosen to maximise profit. In many models, firms do not necessarily own the capital they use. Instead, they may rent this capital from households. The production function for a firm can take the form of a *Cobb-Douglas* function [18], with Output $\propto L^{\alpha}K^{1-\alpha}, \alpha \in (0, 1)$, though this is not the only possible choice [47].

Markets

The markets for labour and goods are assumed to be efficient, and have a tendency to reach equilibria (i.e., wages and prices tend towards values that balance supply and demand). In Keynesian models (such as [136]), mechanisms may be included that inhibit the process of markets reaching perfect equilibria. For example, the frequency at which firms can adjust their prices may be constrained.

Central Bank

A central bank sets economy-wide interest rates. Interest rate decisions are often modelled using a *Taylor Rule*, which assumes that interest rates depend on inflation and unemployment (with high inflation pushing interest rates upwards, and high unemployment pushing interest rates downwards).

Random Shocks

The economy is hit by unpredictable shocks in its variables at each time step. These shocks can have a lasting effect, though it is usually assumed markets would tend towards equilibrium if no subsequent shocks occurred.[2] Shocks are commonly assumed to be drawn from Gaussian distributions with a mean of zero. These shocks make a DSGE *stochastic*. Without them, the economy would remain in a constant state.

[2]Note that in some models, certain shocks can alter the long-term equilibrium.

Other features that may be added to DSGEs include financial sectors, banking/loan markets, housing markets, international trade, and bounded rationality.

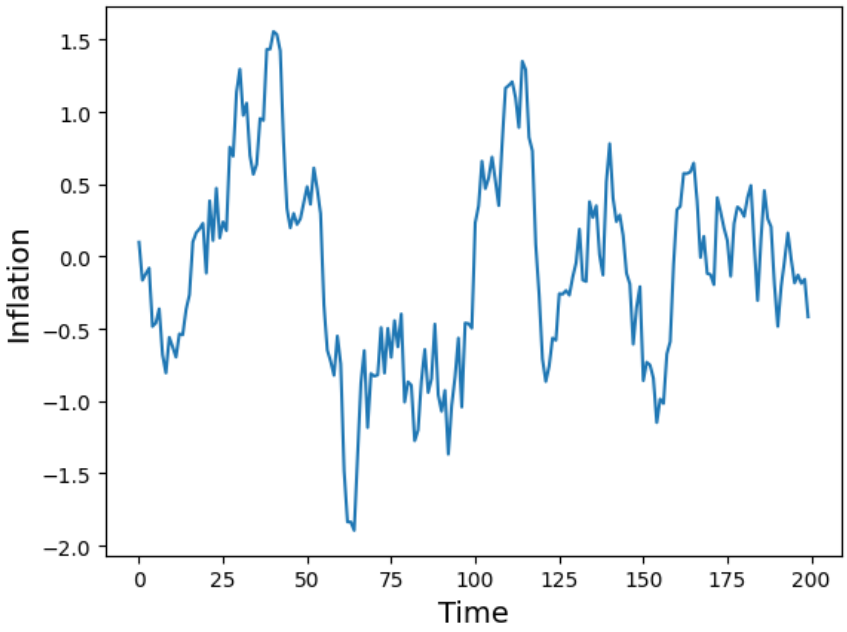


Figure 1.1: **SW DSGE simulation.** Simulation of inflation, generated using the SW model. Inflation varies significantly at each time step, due to the presence of exogenous shocks. Parameter values are listed in Appendix B.1.1.

1.1.3 Steady States and Linearisation

DSGEs possess a steady state, corresponding to demand and supply being in balance across all markets. This is not a steady state probability distribution – it is a fixed-point state, where all variables are constant over time in the absence of shocks. With symmetric shocks added, variables usually oscillate around their steady state values. This behaviour is desirable for several reasons, including the need to make variables statistically stationary, for estimation purposes. However, many economic variables (including GDP) are non-stationary in the real world, with a general upward trend observed over time. Making these variables stationary can involve using differenced variables (e.g., $\Delta\text{GDP}_t = \text{GDP}_t - \text{GDP}_{t-1}$), or using the "output gap". The output gap is commonly defined as the difference between actual GDP and an estimate of the maximum sustainable level of GDP [85], which is zero at equilibrium. Unemployment is commonly assumed to have a stable long-term rate (the "Non-Accelerating Inflation Rate of Unemployment", NAIRU [11]), as is the case for interest rates (the long term "Natural Rate", and the shorter-term "Neutral Rate" of interest [39]).

When a DSGE model is analysed and used to create simulations, it is common to *linearise* (or *log-linearise*) the model around its steady state. When small shocks move variables away from their steady state values, the linearised model makes computing subsequent states more tractable [139], compared to using the nonlinear DSGE equations in their raw form.

1.1.4 Impulse Response Functions

The methods developed in this chapter make use of IRFs. These are commonly used to study DSGEs, vector autoregressions (VARs), other econometric models [89], problems in acoustics [120], signal processing, and control theory [105].

In the context of DSGEs, IRFs describe how the model responds to an unanticipated shock in one or more variables. IRFs can provide insight into the behaviour of the model, including the relative importance of different shocks, and the effect a change in policy may elicit.

DSGE and IRF Definitions

A (discrete-time) DSGE model can be defined using the following equation:

$$X_{t+1} = f(X_t, \varepsilon_{t+1}), \tag{1.4}$$

where X_t is the vector of state variables (e.g. GDP, inflation) at time t, ε_t is the vector of shocks at time t, and f is the vector function describing the evolution of the DSGE's state. Depending on the model, the unit of time may be quarters [134, 69] or months [34]. The initial value of each state variable is equal to its respective steady-state value (representative of macroeconomic equilibrium), meaning X_0 is the fixed point of f under no shocks (i.e., $f(X_0, \mathbf{0}) = X_0$). We assume f is continuous with respect to X, and that if no shocks are present, repeated application of f to any state vector X will lead to convergence towards a steady state:

$$\lim_{n\to\infty} f^n(X, \mathbf{0}) = X_{\text{SS}}, \tag{1.5}$$

where X_{SS} is the steady state. This is equivalent to requiring the DSGE system to converge towards a macroeconomic general equilibrium when no shocks are present. Additionally, if f is parameterised by a vector θ (i.e. $f(\theta, X_t, \varepsilon_{t+1})$), we assume it is continuous and twice differentiable with respect to the parameters.

Let $\text{IRF}(t)$ be the IRF (a vector function). Furthermore, let $\text{IRF}_{ij}(t)$ be a scalar function representing the response of the ith variable to shock j (of magnitude σ_j). Now define the initial values:

$$\text{IRF}_{ij}(0) = \begin{cases} 0 & \text{if } i \neq j, \\ \sigma_j & \text{if } i = j \end{cases} \tag{1.6}$$

and

$$\text{IRF}(t+1) = f(\text{IRF}(t), \mathbf{0}). \tag{1.7}$$

Note that a model is usually said to have a set of impulse response *functions* (i.e., the scalar functions $\text{IRF}_1(t), \text{IRF}_2(t), ...$), rather than a single vector-valued impulse response *function*. Throughout this thesis, we will refer to the IRFs of a given model as separate functions, even though they could instead be considered a single vector function ($\text{IRF}(t)$).

An example of a set of IRFs is shown in Figure 1.2, showing three variables' responses to three different shocks.[3]

[3]These IRFs were generated from the Smets and Wouters DSGE [136]. Not all variables and shocks are shown.

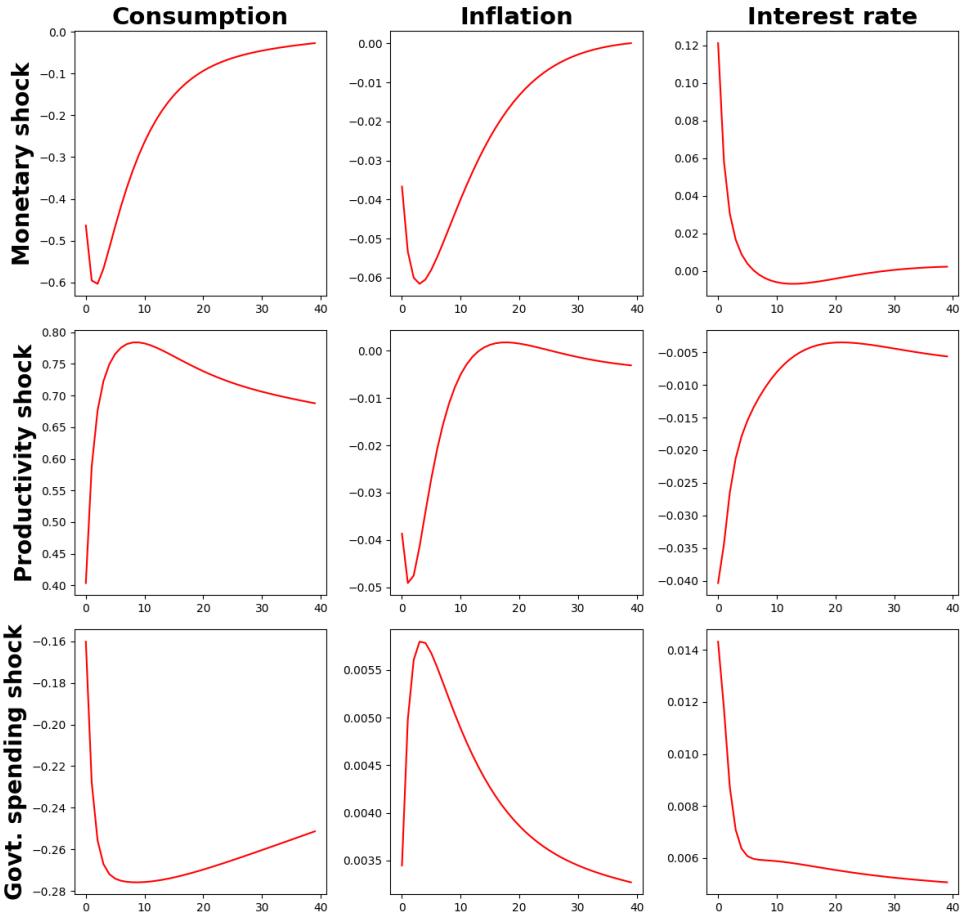


Figure 1.2: **Impulse response functions for SW model.** The row labels indicate the type of shock, and the column labels specify the variables being plotted. (For example, the centre-left plot illustrates how economy-wide consumption evolves after an unanticipated change in productivity.) Forty time periods are shown, with each time step representing one quarter of a year. Parameter values are listed in Appendix B.1.1.

1.1.5 Computational Expense: Generating Time Series vs Computing Impulse Responses

Below, we describe the steps required to compute Jacobians using each method, in order to illustrate the advantage of the IRF method in terms of computational expense. The following notation will be used:

- i is the index for variables. It can take values from set ξ_i.
  - e.g., if there are three variables, then $\xi_i = \{1, 2, 3\}$.

- j is the index for shocks. It can take values from set ξ_j.
  - e.g., if there are three shocks, then $\xi_j = \{1, 2, 3\}$.
- t is the index for the time steps. It can take values from set ξ_t.
  - e.g., if there are T time steps, then $\xi_t = \{1, 2, ..., T\}$.
- m is an index for parameters.
  - e.g., if the parameters are $\theta_1, \theta_2, \theta_3$, then $m \in \{1, 2, 3\}$.
- (ijt) is an element of the Cartesian product of ξ_i, ξ_j and ξ_t, i.e. $(ijt) \in \{(i, j, t) : i \in \xi_i, i \in \xi_i, t \in \xi_t\}$. (ijt) is a scalar.
- J is the Jacobian matrix.
  - The elements of J are $J_{(ijt),m}$
  - (ijt) is the row index, m is the column index.

To generate a set of T-step time series across N_{Endo} endogenous variables:

1. Solve the DSGE at θ.
2. Generate shocks for each of the N_{shocks} exogenous variables subject to shocks at each of the T time steps.
3. Repeat the following N_δ times:
   - For each time step t, feed the new shock vector ε_{t+1} into the model function to generate new values of endogenous variables ($f(X_t, \varepsilon_{t+1})$).

To compute derivatives:

1. Select a random seed, s.
2. Draw a set of T random shocks from that seed, ε_t^s.
3. Simulate the DSGE for T steps with parameter vector θ, using ε_t^s as the shock values.
4. Simulate again (using the same seed, s) at perturbed versions of θ (call them $\theta + \delta_i, \forall i$).
5. **Repeat** over K different seeds, and average for each i to compute the Jacobian matrix $J_{(ij)}$.

The simulations are stochastic, so to estimate the derivative, many time series runs must be generated at the parameter vector θ and perturbed versions of it ($\theta+\delta_i$). The precise number of runs needed to compute accurate SFD derivatives depends on the degree of stochasticity and magnitude of the model's sensitivity to parameter variation (with less sensitivity meaning more runs are required to overcome the "noise" resulting from the stochasticity). It is difficult to determine a priori how many runs are required to obtain an accurate SFD Jacobian, and convergence tests (carried out on either the Jacobian itself, or the eigenvectors of $J^T J$) are needed to assess the accuracy of the estimate.

To generate a T-step sequence of IRFs across $N_{Endo} \times N_{shocks}$ variable-shock combinations:

1. Solve the DSGE at θ.
2. For each of the N_{shocks} shocks, initialise the system based on the first shock, then recursively compute the state of the endogenous variables for T steps (without adding any more shocks into the system).
    - This is run only once for θ, as it generates IRF curves which do not have random variation (unlike simulated time series).

To compute IRF derivatives:

1. Compute the IRF functions for T_{IRF} steps using ε^s as the initial shock value.
2. Simulate again at perturbed versions of θ (call them $\theta + \delta_i, \forall i$).
3. (This yields the IRF Jacobian with elements $J^{\text{IRF}}_{(ij),t}$)

From a computational perspective, the use of IRFs has two advantages when computing derivatives: (1) When generating IRFs, new shocks do not need to be applied at each time step, and (2) IRF generation needs to occur only once for each θ, as opposed to the many runs needed at each θ for stochastic finite differencing on full time series outputs.

A Note on Steady States and RNG Seeds

As discussed in Section 1.1.3, DSGE models usually have a steady state, where the values of all variables are constant. This state is maintained in the absence of shocks [49]. This steady state may be zero, which is often the case when the variables of interest represent deviations from "long-run trends". As shocks are usually symmetric

(i.e., mean-zero), repeatedly running the DSGE at any parameter vector θ will lead to a mean path of $Y_v(t) = 0$, for each variable v. The mean of Y_v as a function of t (i.e., $\langle Y_v(t;\theta)\rangle$) is independent of θ, so any change in the behaviour of a DSGE due to parameter variation may be difficult to detect by simply taking the mean of many runs. The choice of parameters may affect the magnitude of oscillations and correlations between variables – which may make model output differences measurable – but there is an alternative method that better detects the effect of parameter variation. The method is to use the same random seed for each pair of runs at θ and $\theta + \delta$, so a model's response to the same series of shocks can be compared (as is done in [112]). This technique was used when generating results throughout this thesis.

1.2 Method

In this section, we provide a definition of IRFs, and discuss how we will evaluate the success of the IRF method using three different techniques.

1.2.1 Choosing IRF Variables and Shocks

While the analysis of model sloppiness often includes the analysis of all model variables [68], it is possible to consider only a subset of the variables. In this chapter, we will focus on only one variable, Y (representing GDP, or consumption in the case of the Hansen model). When investigating the DSGE models, we treat them as scalar time series generators, taking several parameters as inputs, and being subject to one or more exogenous shocks. In effect, we treat the DSGEs as simulators of a single macroeconomic variable. This choice allows us to use the DSGEs as simple proof-of-concept test cases for our method, without needing to merge and scale a multi-channel time series.

The dominant effect of shocks on the Y variable is contained within the first-order IRFs of variable Y. When defining our χ^2 function (from which we derive the FIM by taking the Hessian), we will not use the IRFs which correspond to the direct effects of shocks on variables other than Y (e.g., the effect of a government spending shock on consumption). Our cost function is, therefore:

$$\chi^2(\theta, \theta^*) = \textbf{const} \times \sum_j \sum_t \left(\frac{\mathrm{IRF}_{Y,j}(\theta, t) - \mathrm{IRF}_{Y,j}(\theta^*, t)}{1} \right)^2, \tag{1.8}$$

where $\text{IRF}_{Y,j}$ is the impulse response of Y to the jth shock, and $\chi^2(\theta, \theta^*)$ measures the quadratic deviation induced by moving from point θ^* in parameter space to point θ. Note that χ^2 is not a measure of error – it is only used to determine the stiff/sloppy eigendirections of the model. The denominator (usually σ_j) has been replaced with 1, as we wish to consider the effect each shock has on Y in absolute terms – that is, if one particular shock has a minimal effect on Y, we do not wish to scale it up. To compute the stiff/sloppy eigendirections of the IRFs, we estimate the Hessian of this cost function, and carry out an eigendecomposition. If our method is successful, these IRF-derived eigenvectors will align with the eigenvectors derived from the SFD method.

1.2.2 Performance Measurement

For each DSGE model, we computed the eigenvectors and eigenvalues using the IRF and SFD methods. We then evaluated the IRF method's performance in several ways. We compared the eigenvalue spectra (to see if the two methods predicted the same sloppy eigenvalue structure), and computed cosine similarity matrices, which show the degree of alignment between IRF and SFD eigenvectors. We also computed the root mean square error (RMSE) induced by perturbing parameters in different IRF eigendirections, to see if the stiff IRF directions caused greater changes in model outputs than sloppy directions did. While all three evaluation methods are useful, our primary success measure is the degree of alignment (cosine similarity) between the eigenvectors computed using the IRF method and those computed using SFD. This is because most applications of computing a model's eigenvectors relate to moving in parameter space in the determined eigendirections, making the accurate calculation of these directions most important.

We examined several DSGE models, the first of which was the Smets & Wouters (SW) model. A detailed description of the DSGE models used in this chapter can be found in Appendix B.1. The default parameter values for each model were taken directly from the "*DSGE_mod*" library[4] [116], which themselves generally came from the papers in which the models were introduced. We used these parameter values as the starting point for our analysis, because these values produce the DSGE behaviours that were investigated in each model's respective study.

[4]An exception was made for the Gali 2015 model, where all three shock variances were set to 0.25^2, instead of zero (which was the *DSGE_mod* default for two of the three shocks). This was done so the DSGE had multiple IRFs per variable, as was the case when the model was first published [55].

Eigenvalue Spectra

If our method is successful, we expect to see a similar pattern when inspecting the ratios between eigenvalues from the conventional (SFD) method and the eigenvalues from the IRF method. However, this is not strictly necessary if the primary goal of this exercise is to identify stiff eigendirections for the sake of exploring parameter space efficiently.

Cosine Similarity Matrix

If the IRF method can accurately compute the stiff and sloppy directions of a DSGE model, we expect the IRF-derived eigenvectors to align with the SFD-computed eigenvectors. To measure alignment between two vectors, we make use of the absolute cosine similarity (CS):

$$C_{ij} = |V_i \cdot U_j| \tag{1.9}$$

where V_i are the SFD-computed eigenvectors, and U_i are the IRF-computed eigenvectors. Both are assumed to have been normalised to unit length. If alignment is perfect, we expect the matrix to be diagonal with $C_{ii} = 1$, and zero elsewhere. We can write this as:

$$C_{ij} = \delta_{ij}. \tag{1.10}$$

Notably, our method can identify the correct eigenvectors, but it may order them incorrectly. For example, it is possible for a surrogate FIM's first eigenvector (i.e., associated with the highest eigenvalue) to be aligned with the model FIM's second eigenvector. This is especially plausible when a model has a small separation between its highest and second-highest eigenvectors (which can occur when the model has two similarly stiff directions, or does not exhibit a sloppy structure at all). If this is the case, we may not observe $C_{1,1} \approx 1$, and instead may find $C_{1,1} \approx 0$ and $C_{1,2} \approx 1$, for example.

To evaluate the significance of the CS results, it is worth considering the CS of two random vectors. Table 1.1 shows the expected values of the CS of two random vectors in p-dimensional space. These values represent what we would observe in our matrices if our IRF method was no better at predicting eigendirections than a method generating random vectors. These values offer a useful way to assess whether our method has successfully identified stiff and sloppy directions (at least in an approximate form), or has completely failed to fulfil this objective.

Parameter Count (p)	Mean Absolute CS
2	0.63
3	0.50
4	0.42
5	0.37
6	0.34
7	0.31
8	0.29
9	0.27
10	0.26
11	0.24
12	0.23
13	0.23
14	0.22
15	0.21
16	0.20
17	0.19
18	0.19
19	0.18
20	0.18

Table 1.1: **Mean absolute cosine similarity between two random vectors** in p dimensions. If our IRF method successfully reproduces the SFD eigenvectors, we expect to see cosine similarities significantly greater than the cosine similarity between random vectors. Each row was computed using 50,000 random pairs of vectors.

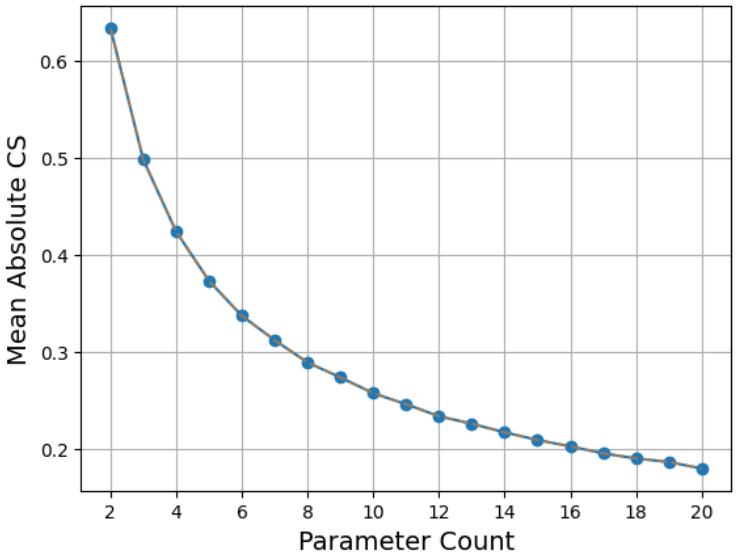


Figure 1.3: **Mean absolute cosine similarity between two random vectors** in p dimensions. Each point was computed using 50,000 random pairs of vectors.

1.2.3 First Test Case: Smets & Wouters Model

The Smets & Wouters model is a DSGE originally designed to simulate the Euro Area economy[5] [135]. A 2007 paper presented an adapted version of the model, designed to simulate the US economy [136]. We will make use of the 2007 model, which will henceforth be referred to as the "SW model" (or simply "SW").

SW Model Features

In the SW model, households supply labour to firms (via employment), and they use their earnings to purchase goods for consumption and invest in capital. Firms produce goods using labour and capital as inputs. They set their sale price, and sell to households. Additionally, a central bank sets the interest rate faced by firms when borrowing capital from households.

Random shocks occur at each time period, including unexpected advancements in technological capability (affecting the production of goods per unit of labour and capital), unanticipated shocks in government spending, and unanticipated changes in household preferences (including their degree of preference for leisure over labour). The variables of the SW model include GDP or output gap (Y), consumption (C), inflation (π), interest rate (r), capital stock (K), and labour supply (N, number of full-time equivalent workers). A more detailed description of the SW model is provided in Appendix B.1.

We use the SW model for our first demonstration of the IRF method, as it is a widely studied model of significant complexity. The successful application of the IRF method on the SW model (demonstrated below) indicates the IRF method can be applied usefully to models used in the real world (as opposed to simple toy models). The IRF method is applied to a wider range of models in Section 1.4, later in this thesis.

1.2.4 Other DSGE Test Cases

In addition to the SW model, we applied the IRF technique to more DSGE models, and a vector autoregression (VAR). The additional models analysed are listed in Table 1.2. The code for simulating each model was sourced from the *DSGE_mod* GitHub repository [116].

[5] "Euro Area": the combined economy of the EU member states which use the Euro as their official currency. This excludes non-EU states that use the Euro as their currency, such as Montenegro.

Model Paper	Description	Parameters Used	Shocks
Aguiar & Gopinath 2007 [1]	A VAR, used to identify the economic shocks which drive business cycles in developing economies	5	2
Gali 2015, chapter 8 [55]	A New Keynesian DSGE model of small open economies	5	3
Schmitt-Grohe & Uribe (SGU) 2004 [126]	A stochastic neoclassical growth model, created to demonstrate that under certain conditions, model coefficients need not be varied when stochasticity is introduced to a deterministic growth model	5	1
An RBC model	A DSGE model without market frictions, used to study the effect of government spending and productivity shocks. Inspiration for this simple model comes from [86] and [123]	4	2
Hansen 1985 [69]	A DSGE with indivisible labour (workers cannot be part-time), which seeks to explain high fluctuations in unemployment	5	1

Table 1.2: **Models analysed using the IRF method** in addition to the SW model, showing the number of parameters varied during sloppiness computation (i.e., the number of eigenvectors computed).

1.3 Results

The results of applying the IRF method to the SW model are presented and discussed below, followed by the results for the other DSGE models. Jacobians were computed ten times using SFD, then averaged before computing $J^T J$. The eigenvalues and eigenvectors were then computed from this matrix. The eigenvalues were normalised (so $\lambda_1 = 1$), and these values were used to sort the eigenvectors from stiffest to sloppiest.

1.3.1 SW Eigenvalue Spectra

Figure 1.4 demonstrates that the IRF method successfully recovered (a) the total spread of eigenvalues that indicates sloppiness, and (b) the relative ratio of eigenvalues seen when the SFD method for computing sloppiness was applied.

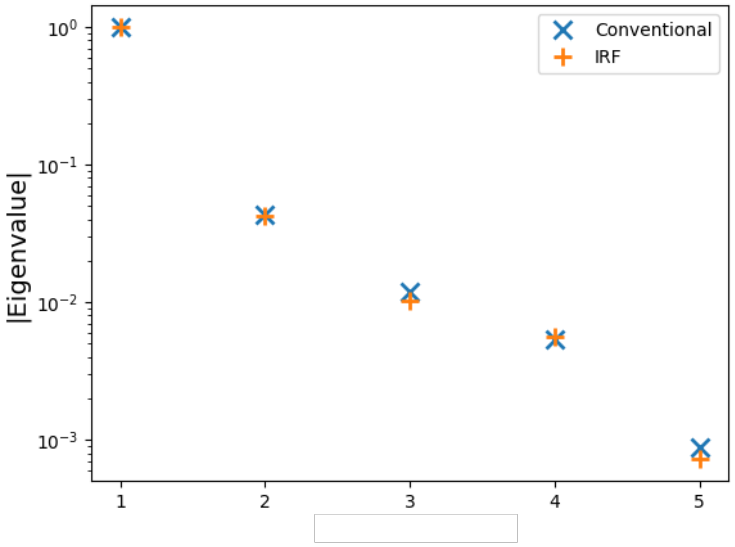


Figure 1.4: **SW eigenvalue spectrum**. Normalised eigenvalue spectrum for the SW model, derived from the SFD and IRF methods. The close alignment indicates the IRF method has successfully identified the model's true sensitivity to different perturbations. Parameter values used are listed in Appendix B.1.1.

1.3.2 SW Cosine Similarity

The CS matrix for the SW model is nearly diagonal, as shown in Figure 1.5. This indicates a positive outcome: in most cases, the IRF method successfully identified the same stiff and sloppy eigenvectors as those computed using the SFD method.

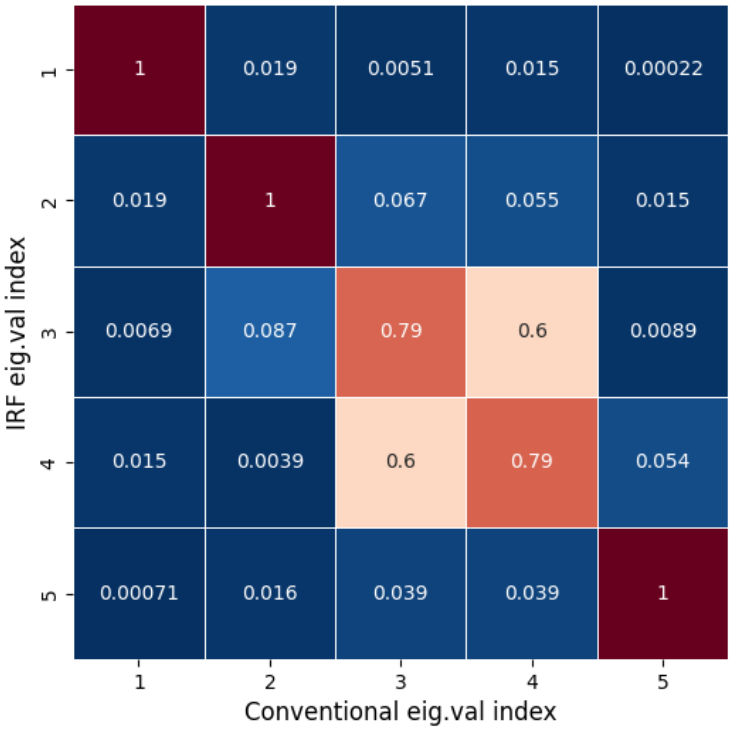


Figure 1.5: **SW CS matrix.** Conventional SFD-derived eigenvectors show a high degree of alignment with IRF-derived eigenvectors. The third and fourth eigenvectors are less closely aligned, indicating some eigenvector "mixing" took place, where the third IRF eigenvector is a linear combination of the third and fourth SFD eigenvectors. Parameter values are listed in Appendix B.1.1.

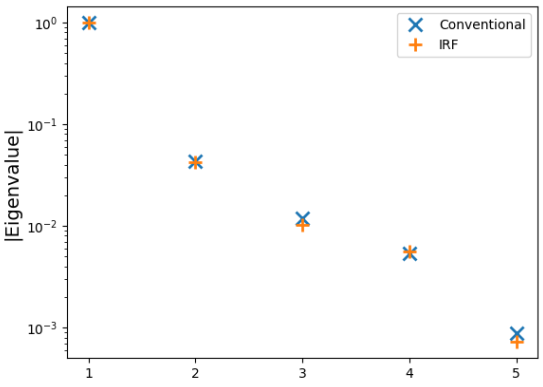
1.3.3 Measuring Deviation

Suppose we run our model using a length-p parameter vector θ, and suppose it generates a time series, Y. When parameters are perturbed in the stiffest direction ($\theta + sv_1$, where s is a small positive scalar), we expect the output (Y_1') to diverge considerably from the original output. Conversely, perturbing the parameters in the sloppiest direction (v_p) should lead to an output (Y_p') very similar to the original. The eigenvectors with index between 1 and p (i.e., eigenvectors $2, ..., (p-1)$) should have an intermediate effect. This behaviour is illustrated in Figure 1.4, which gives a qualitative indication that stiff and sloppy eigenvectors have been successfully identified. As a quantitative metric, we use Root Mean Square Error (RMSE) to measure the change in Y resulting from a perturbation. We computed the RMSE between pairs of outputs, averaged over K simulations. We expect $\text{RMSE}(Y, Y_1)$ to greatly exceed $\text{RMSE}(Y, Y_p)$. This provides a very simple comparison metric and provides insight into our method's performance. As shown in Table 1.3, we did see

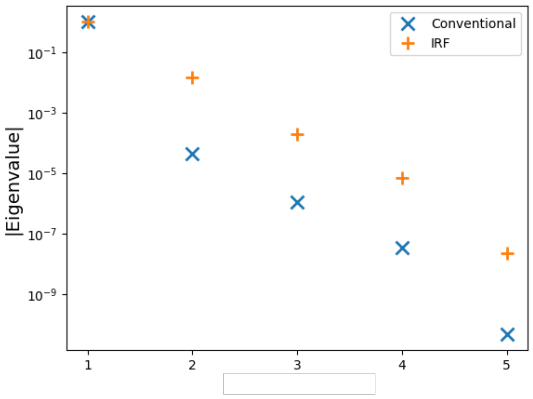
$\text{RMSE}(Y, Y_1) > \text{RMSE}(Y, Y_2) > \text{RMSE}(Y, Y_p)$ for the SW model.

1.4 Results for Other DSGE Models

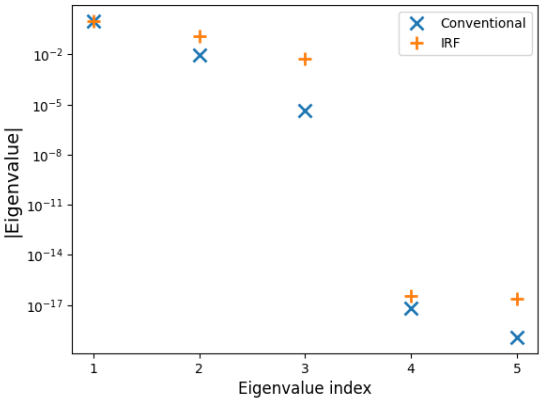
In Figures 1.6 and 1.7, we display the eigenvalue spectra and CS matrices for the other models studied. Additionally, we provide the RMSE values, and illustrate how perturbations in stiff and sloppy directions affects the time series output of a model (Figure 1.4).



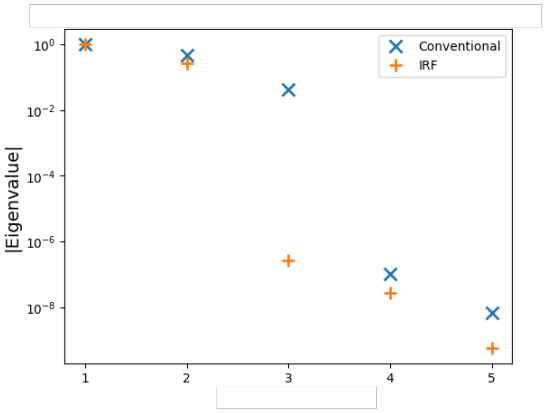
(a) Smets & Wouters 2007



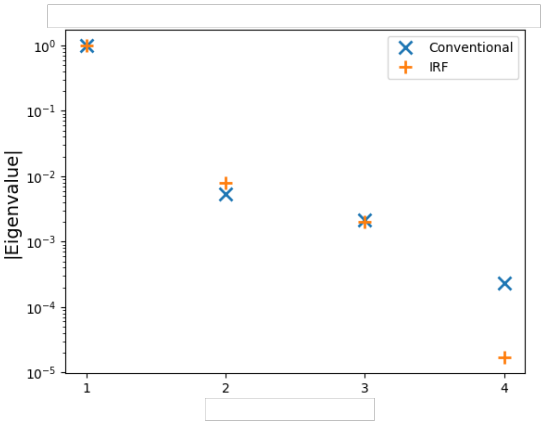
(b) Aguiar & Gopinath 2007



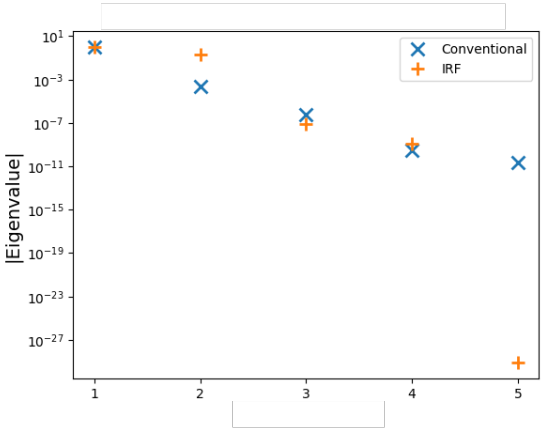
(c) Gali 2015



(d) SGU 2004

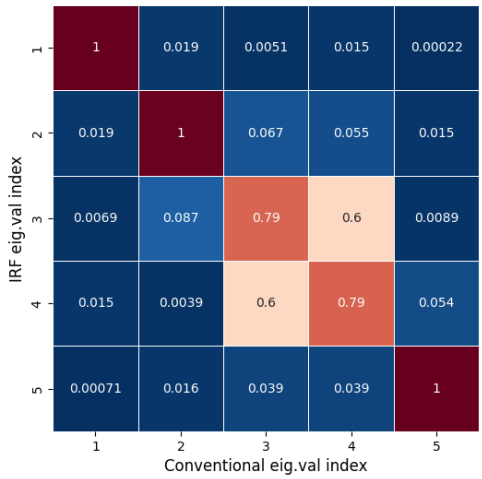


(e) RBC model

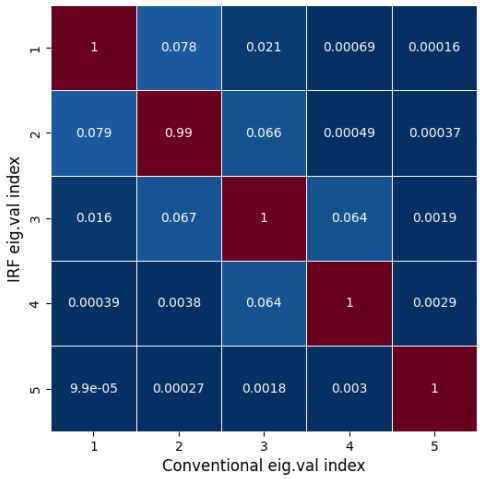


(f) Hansen 1985

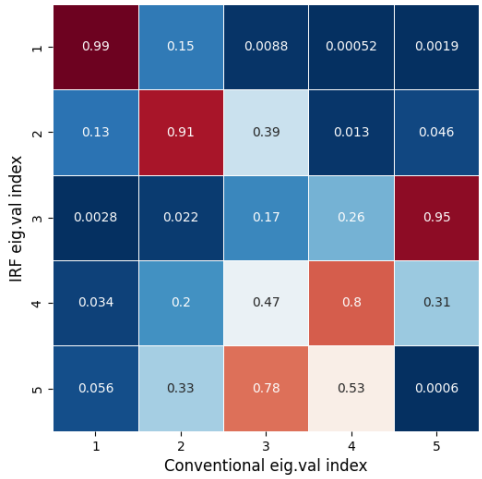
Figure 1.6: **Eigenvalue spectra for DSGE models**. Close alignment between all but the sloppiest eigenvalue pairs was observed in most models, with the exceptions being the Aguiar & Gopinath, and SGU models. This suggests the IRF methods has been able to recover the "structure" (relative sloppiness) of the models, in most cases. Parameter values are listed in Appendix B.1.



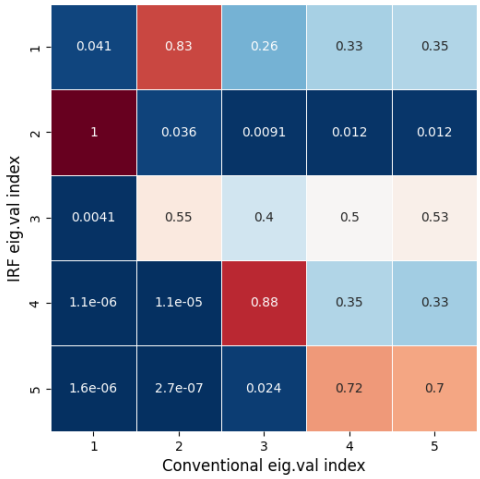
(a) Smets & Wouters 2007



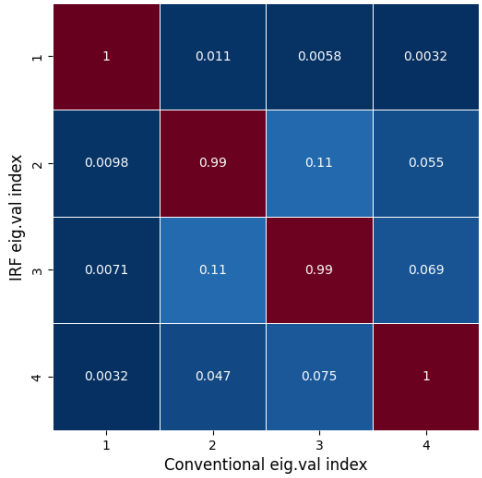
(b) Aguiar & Gopinath 2007



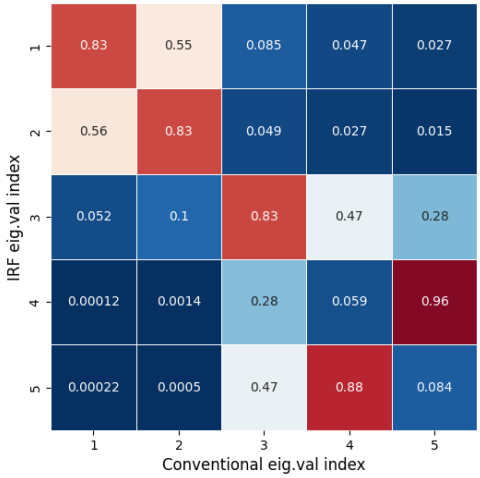
(c) Gali 2015



(d) SGU 2004



(e) RBC model



(f) Hansen 1985

Figure 1.7: **Cosine similarity matrices.** Strong alignment between SFD and IRF eigenvectors was observed in most cases. However, some eigenvector swapping was observed, especially in the case of the Gali 2015 model. Parameter values are listed in Appendix B.1.

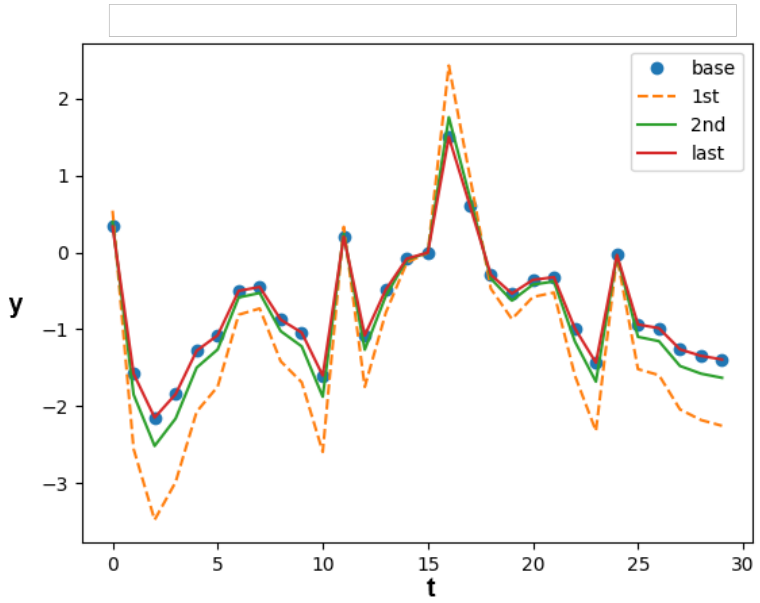


Figure 1.8: **Deviation plot showing simulated time series**, using the Gali 2015 model. Perturbing parameters in different IRF-derived eigendirections ("1st" is stiffest eigendirection, "last" is sloppiest) led to different magnitudes of deviation from the baseline (non-perturbed) simulation. Perturbing in the stiffest direction (orange line) leads to the greatest change, while perturbing in the sloppiest direction (red line) leads to virtually no visible change compared to the baseline. This indicated the IRF-derived eigenvectors aligned well with true stiff/sloppy eigendirections of the model. Parameter values are listed in Appendix B.1.3.

Model	RMSE(1st)	RMSE(2nd)	RMSE(last)
Smets & Wouters 2007	0.5636	0.1141	0.0201
Aguiar & Gopinath 2007	0.2590	0.1529	0.0005
Gali 2015	0.1967	0.0635	0.0022
SGU 2004	0.1203	0.1339	0.0689
RBC model	2.5076	0.1763	0.0267
Hansen 1985	0.7318	0.2360	0.0200

Table 1.3: **RMSE values for DSGEs**, resulting from perturbations in the direction of the two stiffest IRF eigendirections ("1st", "2nd") and the sloppiest eigendirection ("last"). For all models, we see RMSE(1st) > RMSE(last), as expected for stiff vs sloppy perturbations. However, in the case of the SGU model, we saw RMSE(1st) < RMSE(2nd), which can be explained by the eigenvector swapping we saw in Figure 1.7 for the SGU model. Ten trials were used to compute each RMSE value.

1.4.1 Cosine Similarity Matrices and Eigenvalue Spectra

In Figure 1.7, we generally see high values on the diagonal elements of the CS matrices. The highest values (i.e., the best alignment) were generally seen between the first two pairs of eigenvectors (top-left of the matrices). We saw this pattern throughout this thesis: identifying the stiffest eigenvectors tends to be more straightforward than identifying the sloppiest.[6]

However, for two models the leading eigenvectors did not align well: the SGU 2004 model, and the Gali 2015 model. In the case of SGU, the first and second eigenvectors were identified relatively accurately ($C_{12} = 1.00$, $C_{21} = 0.83$), but their order was swapped. Figure 1.6 provides a possible reason for this: the first and second eigenvalues of the SGU model are similar in magnitude. This resemblance suggests that the two corresponding eigendirections had similar degrees of stiffness and that using an imperfect method of estimating eigenvectors/values (i.e. the IRF method) had led to the second-lower magnitude eigenvalue being estimated as the greatest. This possibility was corroborated by two observations. Figure 1.6 shows the first and second eigenvalues were of similar magnitude, indicating the stiffness of the two respective eigenvectors was similar. Additionally Table 1.3 shows that the RMSE values for the first and second eigenvectors were similar, indicating that perturbations by each of the two eigenvectors had a similar impact.

With regard to the Gali model, the IRF method performed less effectively than when applied to the other models. The CS for the first and second eigenvectors was around 0.65, higher than the CS of random vectors in five dimensions (0.37). However, this level of alignment was significantly less than that seen in our other models. It appears that the first and second IRF-derived eigenvectors were each a linear combination of the two leading eigenvectors derived using SFD. This phenomenon can be seen by inspecting the upper-left 2×2 square of the matrix, which shows the first IRF eigenvectors were best aligned with the first two SFD eigenvectors (and vice versa). Unlike the case of the SGU results, the leading two eigenvalues were dissimilar in value, and the RMSE values differed substantially. For this reason, it was unclear why the IRF method produced "mixed" eigenvectors (as opposed to the "swapped" eigenvectors seen in the SGU case).

[6]This phenomenon is discussed further in section 2.8.2

	Aguira	Gali 2015	RBC	SGU	SW	Hansen
Simulation Timesteps	120	200	200	200	200	200
DSGE Sim Time (sec)	1.4	1.7	3.3	0.9	18.1	1.3
SFD Hessian Time (sec)	156	182	293.1	94.9	1990.8	144.5
IRF Hessian Time (sec)	15	18.2	29.5	9.4	198	14.4
Speed-up Factor (SFD/IRF)	10.4	10	9.9	10.1	10.1	10

Table 1.4: Time comparison for eigendirections derivation using the SFD and IRF methods. SFD Jacobians were computed using central differencing, with perturbation magnitude being approximately 10% of parameter values. SFD Jacobians were produced ten times, then averaged to compute a single J matrix. Variation in the speed-up factor (slightly above or below ten) was the result of small time differences in overheads required for eigendecompositions (SFD and IRF), matrix averaging (SFD only), and other non-simulation processes required for one or both methods.

1.5 Discussion

The IRF method was generally successful in identifying the sloppy and stiff eigendirections of most of the DSGE models investigated. Where directions were not accurately identified, this generally coincided with a pair of eigenvalues being of very similar magnitude, which allowed for eigenvector orderings to be swapped, or two IRF eigenvectors to be linear combinations of SFD eigenvectors. With the exception of the SGU model, perturbing parameters in the directions that the IRF method identified as stiff led to greater deviation (as measured by RMSE) than perturbations in sloppy directions. This serves as further evidence that the IRF method is capable of identifying stiff and sloppy eigendirections.

The cases where the IRF method failed to accurately estimate eigendirections (or their ordering) generally corresponded to two features: eigenvalues being close in magnitude and easily swapped (as discussed previously), and a small number of shocks. The weakest results were observed for the SGU and Hansen models, both of which have only a single shock. This reduces the number of IRFs for these models (given that each IRF corresponds to a variable-shock combination), meaning the IRF method has less information from which to estimate the stiff and sloppy eigendirections.

The time taken to compute Hessians via the IRF and SFD methods is shown in Table 1.4. In all cases, IRF Hessians were around ten times faster to compute compared to SFD Hessians. If SFD Hessians could be computed in a matter of seconds, the speed-up would not be particularly useful. However, because the SFD

Hessians were often very slow to compute (taking between two and thirty minutes, depending on the complexity of the DSGE) the IRF speed-up represents a significant advantage. Additionally, it may be possible to further decrease the time taken for IRF Hessians: in our tests, we generated the simulations and IRFs at the same time, but it would be possible to speed up the process by generating the IRFs alone. Additionally, the IRF time series length was the same as the simulation time series length. A further reduction in compute time may be achievable by using shorter IRF lengths (although this would require further experimentation, to establish if it causes any loss in accuracy).

1.6 Chapter 1 Conclusion

In this chapter, we presented a method for using IRFs to determine the stiff and sloppy eigendirections of DSGE models. This method was applied to a range of models, including a small-scale RBC, and a larger model which is the basis of DSGEs used in the ECB. Three methods for evaluating success were applied – eigenvalue spectra comparison, measuring alignment between IRF and SFD eigenvectors, and computing the deviation in model output which results from perturbation in IRF-derived eigenvectors. When comparing eigenvalue spectra, we found that the IRF and SFD spectra often mismatched, but this did not necessarily lead to poor alignment and RMSE results. Perturbations along IRF-derived eigenvectors induced a much greater RMSE than perturbing in the sloppiest direction (the difference sometimes being a factor of one hundred).

Alignment between eigenvector pairs was generally strong, being in excess of 0.8 for the stiffest eigenvectors in all but one case (for comparison, average absolute CS between two random vectors in the same number of dimensions is around 0.37, on average). As movement along stiff and sloppy eigenvectors is the primary process in most applications of the principles of sloppiness, this is the result provides a justification for considering the IRF method to be successful, for the purposes of deriving stiff directions for parameter space exploration, and sloppy directions for model reduction.

Chapter 2

Surrogate Modelling: Mimicking Complex Systems with Neural Networks

Computing model sloppiness using IRFs was demonstrated successfully in Chapter 1. However, it is desirable to have methods to reliably compute sloppiness that can be applied to a broader range of models, for which IRFs may not exist. This chapter develops a novel method of doing this, involving the use of surrogate models (also known as metamodels) in place of the original models. To make the computation of the stiff and sloppy eigenvectors reliable, we seek to create surrogates that are both differentiable and deterministic, so we can avoid the issues associated with stochastic finite differencing discussed earlier in this thesis.

Our results show it is possible to compute the stiff and sloppy eigendirections for a number of models considered. Despite the simplicity of feedforward neural networks (FNNs), we find they can act as useful surrogates when we seek to compute the eigendirections. We also explore why accurately estimating sloppy directions is often more difficult than estimating stiff ones when using both surrogate methods and conventional SFD. Finally, we discuss the observation that a surrogate may be incapable of accurately emulating a target model – i.e., the surrogate's outputs look very different from those of the model – but may still possess parameter sensitivities similar to those of the model, making identification of stiff/sloppy eigendirections possible using the surrogate.

2.1 Literature Review: Surrogate Modelling

Surrogate models are computational systems that approximate the behaviour of more complex or computationally expensive models. They can be used to study the system they approximate, which may be advantageous if the function is slow or complex to directly simulate [50]. They have been used in numerous fields, including computational fluid dynamics [75, 56, 43], structural engineering [161], materials science [115], and human physiology [100, 91].

2.1.1 History

Experiments involving physical processes can be expensive, making it desirable to develop machines that can quickly simulate and analyse such processes. To this end, computers have been used since the early 1900s. Notable examples include the estimation of bomb trajectories during the Second World War using the Norden bombsight (an analogue computer, prototyped in 1924 [80]), and using digital punched-card machines to model nuclear explosions [57, 33]. These simulators were built by making use of knowledge of the underlying physics of processes (e.g., power laws relating air-resistance to velocity, or the Navier-Stokes equations for other problems in aerodynamics). Such a simulator can be considered a surrogate model, with some form of the relevant laws of physics embedded.

In the 1950s, a form of surrogate modelling which does not require the embedding of laws of physics emerged, named the Response Surface Methodology (RSM) [23]. This technique typically involves fitting a polynomial to model the relationship between a dependent variable and one or more independent variables [62], allowing interpolation between known data points. Nonlinear terms are allowed, including terms representing interactions between variables [9]. Later, the method of Kriging (also known as Gaussian Process Regression) was developed for use in the field of geostatistics [91]. A major difference between Kriging and RSM was the incorporation of nondeterminism in a way that allowed the quantification of uncertainty during interpolation [9, 128].

In the later decades of the 20th century, the increase in computational power and data availability made the use of machine learning techniques practical. Creating (and training) ANNs and other machine learning models became feasible, allowing for the creation of fast and accurate surrogates which could handle complex, high-dimensional data [41]. Since then, the use of advanced machine learning methods to create surrogates has become widespread throughout science and engineering [63].

2.1.2 Challenges in Surrogate Modelling

While surrogate modelling is a relatively mature field, with a wide range of methods available to produce model emulators, challenges persist in the creation of good surrogates. A trade-off exists between simplicity and accuracy: a simple model (such as a polynomial response surface) can have the advantage of being simple to calibrate and easy to interpret, but may be incapable of emulating models with a high degree of nonlinearity [17], such as ABMs [71]. Conversely, advanced models – such as neural networks – can replicate complex behaviours [141], but require computationally expensive training [77, 19]. Additionally, the relationships between variables in such "black box" models often lack interpretability, although techniques exist which attempt to derive interpretations [97, 122].

Data availability may also be a problem, as neural networks are highly vulnerable to overfitting when training data is limited in quantity [17, 141]. Additionally, data coverage can be a critical issue: a surrogate trained on data in one region of parameter space may generalise very poorly when applied to data outside this region [8]. This extrapolation problem is an inherent flaw with all surrogate methods, and must be dealt with by either (a) testing out-of-sample predictions thoroughly to build confidence in the surrogate's ability to generalise, or (b) making use of knowledge of the dynamics of the process being modelled [51].[1] Moreover, when a surrogate is applied to new data inside the training region (say, points within $[0, 1]^N$, where N is the number of independent variables), complications may arise. That is, if the model has many variables, the curse of dimensionality may make generalisation *within* the training region extremely difficult, as random points in high-dimensional space are generally distant [19].

For our use-case, the most significant challenges when creating surrogates are (a) generating training sets with good coverage of high-dimensional parameter spaces, and (b) creating surrogates which can capture an economic model's parameter sensitivities, despite the presence of nondeterminism in that model. Additionally, given that our motivation is to replace the (often very slow) process of SFD computations, validating our surrogates' predictions against SFD is a computational challenge, necessary because SFD Jacobians must be computed for the sake of comparison. Indeed, the computation of SFD Jacobians for validation was found to be the greatest challenge of the whole surrogate creation and evaluation process.

[1] e.g., if a system is known to have linear behaviour everywhere in parameter space, a response surface surrogate trained in one region of parameter space will perform well in other regions too.

2.2 Defining a Surrogate

Assume we have a economic model represented by a function $f : X \to Y$. A function f' can be described as a surrogate for f if $f'(x) \approx f(x)$, for all $x \in X$ (or a sufficiently large subset of X, where "sufficiently large" depends on the application).

Surrogates can be variously classified, such as by the type of output they can produce. Let us define several classes of surrogates in the following way:

- A **Deterministic Surrogate** is a function f_D which can only produce a single output, y, for any given input x.[2]
- A **Deterministic Summarising Surrogate** is a deterministic function f_{DS} which predicts a small set of summary metrics describing distribution of outputs of f for a given x.
- A **Stochastic Generative Surrogate** is a nondeterministic function f_G from which many outputs can be generated using a single input x. For a time series model, this means f_G can produce many different time series when run repeatedly on a single parameter vector x.

While deterministic surrogates (f_D) cannot emulate the stochasticity of a nondeterministic model, they may still be capable of capturing the model's parameter sensitivity. This ability could arise because (a) the stochastic component of the model is relatively small (e.g., the model's outputs oscillate around some mean path, such as the Agent-based SIR defined later), or (b) the surrogate's outputs differ greatly from the model's outputs, but both outputs show similar sensitivities to variations in parameters. Specifically, the stiff/sloppy eigenvectors may be similar, even though the surrogate cannot accurately predict the output of the model. The former case was observed in several models (see results section below), and the latter was observed in other models (including the Brock-Hommes model, as discussed in section 2.8.3).

Generative surrogates can incorporate nondeterminism, inherently present in many economic models. A surrogate capable of generating nondeterministic outputs implicitly contains an internal probability distribution [157], from which draws are made to create output realisations. Conversely, a deterministic summarising surrogate outputs the parameters of the distribution itself (or some parameters which uniquely define the distribution[3]), without producing realisations (draws) from that distribution. A

[2]Note that x and y may be vectors, e.g., $x \in \mathbb{R}^M$ and $y \in \mathbb{R}^N$.

[3]e.g., the mean and standard deviation of a Gaussian distribution.

generative surrogate is effectively an emulator, while a probabilistic surrogate yields statistical information on the likelihood of different possible outcomes. A fuller discussion of how one may determine, a priori, that a deterministic surrogate will likely be insufficient, is provided in Section 3.5.1. This follows a discussion of the results generated by nondeterministic surrogates we trial.

Although deterministic surrogates can "generate" new outputs, we reserve the term "generative surrogate" (or "generative model") to refer to models which can stochastically produce many different outputs (where each output is a time series), when provided with a parameter vector. This contrasts with deterministic surrogates, which can only produce a single output time series when provided with a parameter vector. The definition and properties of generative models is discussed further in section Section 3.1.

In this chapter, we develop and test deterministic surrogates. Chapter 3 of this thesis develops methods to create probabilistic surrogates.

2.2.1 Surrogate Methods

Surrogates can also be classified by the method they use to emulate the target model. In their 2022 survey of surrogate modelling papers, Kudela & Rudel [91] identify five main categories of surrogate: response surfaces and linear regression, Kriging, radial basis functions, artificial neural networks, and support vector regression. An additional, recently developed method for constructing surrogate models is Equation Learning (EQL) [28], which can be used to create dynamical systems models from empirical data, or from the outputs of other mathematical models to create surrogates. EQL has been used to create surrogates for ABMs using differential equations [111], including the SIR model of disease transmission used in this chapter as a test case.

In this thesis, we have chosen to use ANNs as surrogates. The models we study exhibit a large degree of nonlinearity, which is better modelled by the flexibility of ANNs, especially compared to linear and quasi-linear models, such as response surfaces and regressions [38]. ANNs can be scaled for use with high-dimensional input and output spaces [94, 158], which is necessary for the models we study, which produce one hundred or more outputs, in the form of time series. Additionally, complex interactions between variables can be discovered using ANNs [42], which is crucial for sloppiness analysis, as we are often seeking stiff/sloppy parameter combinations, rather than investigating the effect of parameters individually.

2.2.2 Sampling Plan

When a function takes a high-dimensional parameter vector as an input, a very large number of samples may be needed to achieve good coverage of the parameter space – leading to the so-called "curse of dimensionality" [19, 93]. For models with complicated phase spaces – especially where the model's behavioural changes are highly nonlinear – wide sampling coverage is important when building a surrogate, if one wishes for that surrogate to replicate the behaviour of the model throughout the allowed parameter ranges [83]. Depending on the situation, simple random sampling may be sufficient to create a good training set. However, relatively straightforward strategies exist that can greatly improve coverage, including stratified sampling, orthogonal sampling, and Latin hypercube sampling [107].

The most straightforward approach is "simple random" sampling, which involves drawing points from a multivariate uniform distribution across all parameters [51]. All points in parameter space are equally likely to be sampled, and the algorithm is computationally straightforward to implement. However, achieving good coverage – especially in high-dimensional parameter spaces – can be difficult, requiring a large number of samples [107]. To improve coverage, the parameter space can be divided into a large number of sub-regions ("strata"), with these sub-regions are equally likely to be sampled. This is known as "stratified sampling", and sub-regions do not necessarily need to be of the same size [132]. In regions of parameter space where model behaviour is highly sensitive parameter variation, these sub-regions can be subdivided further, to ensure denser sampling in important regions.

Latin Hypercube sampling (LHS) is a method designed to improve coverage, especially in high-dimensional spaces [107]. LHS provides better uniformity across dimensions, by breaking each parameter dimension into n intervals, and generating samples such that every interval contains one point. As a result, clustering within dimensions can be avoided [107]. However, LHS does not guarantee clustering does not take place within the multidimensional parameter space – i.e., for sampled points θ, component θ_i may be significantly correlated with θ_j [142]. To address this weakness, orthogonal sampling has been developed to reduce possible, inadvertent correlations between parameter components [142]. Orthogonal sampling ensures the distribution of θ_i values is orthogonal (i.e., uncorrelated) with the distribution of θ_j [107].

In the experiments we carried out for this thesis, we trialled LHS and simple random sampling. We found that there was no noticeable difference in performance

between the two, and modifying the ANN structure was more impactful. The additional computational cost of using LHS or orthogonal sampling did not seem justifiable, given that we were already facing the computational challenge of computing SFD Jacobians.

2.3 Neural Network Architecture

ANNs are computational graphs, with nodes and edges. Typically, the nodes take scalar real values, and the edges represent relationships between nodes. In this chapter, we primarily use a simple "fully connected" FNN structure. The networks were implemented using the Keras software library [35].

2.3.1 Fully Connected FNNs

A fully connected FNN comprises several layers, each with a set of nodes. FNNs are used primarily for *regression* tasks, involving mapping inputs (independent variables, also called "labels") to outputs (dependent variables). These networks are trained using supervised learning: the network is sequentially given examples – input/output combinations – and induced to learn values for each node, such that unseen inputs will be mapped to their corresponding (also unseen) outputs.

There are three main types of layer in an FNN: the input layer, consisting of nodes that take the values of each independent variable; the output layer, containing nodes that take the values of the dependent variables; and hidden layers, which consist of intermediate nodes that sit between the input and output layers. The number of nodes in the input and output layers are determined by the number of independent and dependent variables. The number of nodes in each hidden layer and the number of hidden layers can be adjusted depending on the task at hand (and some experimentation may be required to determine the appropriate structure).

A simple FNN is illustrated in Figure 2.1.

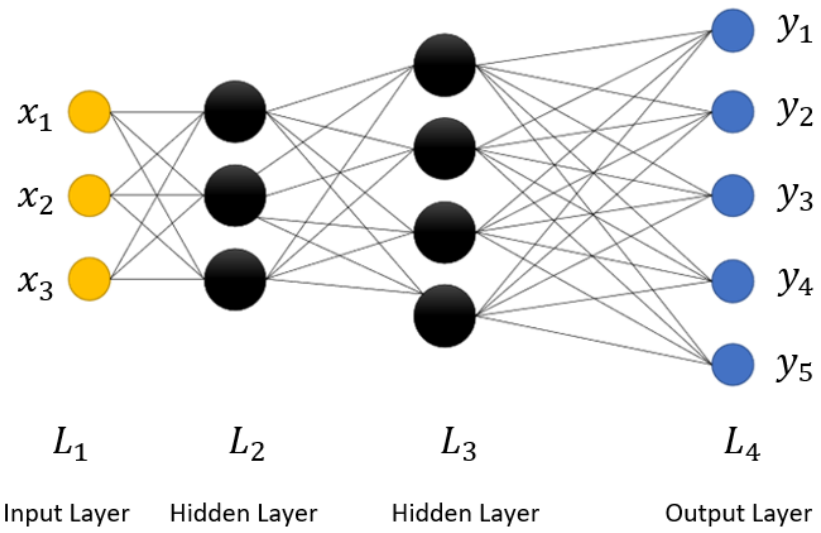


Figure 2.1: **Basic FNN structure**. A network with two hidden layers is shown. FNNs can be constructed with more hidden layers, and hidden layers with many more nodes.

Consider an FNN with four layers: an *input layer* (L_1), two *hidden layers* (L_2, L_3), and an *output layer* (L_4). The nodes in L_1 (which can be represented by a vector, V_1) take the values of a vector of independent variables. Each node in L_2 takes a value equal to a function of every node in the previous layer. Specifically, a layer-two node's value is given by:

$$V_{2,i} = f_a\left(\sum_{j=1}^{N_1}(W_{2,j}V_{1,j} + B_{2,j})\right), \tag{2.1}$$

where $W_{2,j}$ is the *weight* of the jth node of the first layer w.r.t. $V_{2,i}$ and $B_{2,j}$ is its additive *bias* term. After the weight and bias operations are applied to all nodes of layer L_1, the resulting scalar value is fed through an *activation function*, f_a to get the value for $V_{2,i}$. The values of the nodes in L_3 relate to L_2 in the same way L_2 did to the input layer – that is, weights, biases and an activation function are applied to every node of L_2 to determine the value of each node in L_3. The output layer relates to the final hidden layer in the same way. The function f_a is present to create nonlinearity within the network, without which the FNN would be capable of emulating only linear models. Common activation functions include:

- Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$
- Rectified linear unit (ReLU): $\text{ReLU}(x) = \max(0, x)$

- Leaky ReLU: $\text{LeakyReLU}(x) = \max(ax, x), 0 < a \ll 1$
- Hyperbolic tangent: $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Softmax: $\text{Softmax}(x_i) = e^{x_i} / \sum_j e^{x_j}$
- Exponential linear unit: $\text{ELU}(x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$

Each activation function is most suitable for specific tasks. For example, sigmoid functions are commonly used for binary classification tasks. Especially for regression tasks other than classification, it is common to find experimentation necessary to determine the ideal activation function(s). For the FNNs we made use of in this chapter, ReLU and Leaky ReLU activations were found most effective.

Independent and Dependant Variables

We wish to create a surrogate mapping a parameter vector of length P to a full time series of length T. Therefore, it is appropriate for us to use a FNN with P nodes in the input layer, and T nodes in the output layer – or if the time series is multi-channel, $T \times$ (number of channels) output nodes.

2.4 Our Surrogate Method

The steps involved in producing our deterministic surrogates can be summarised as follows.

1. Sample from the parameter space numerous (N) times, producing a set of parameter vectors $\Theta = \{\Theta^1, \Theta^2, \ldots, \Theta^N\}$.
2. For each point in parameter space Θ^k, run the model and save its output, G^k.
    - Note that G^k is the full output(s) resulting from applying the original model to the parameter vector Θ^k. In a macroeconomic model, this could be a multivariate time series (e.g., GDP, inflation, unemployment tracked over time).
3. Train the surrogate network via supervised learning, with input Θ^k corresponding to output G^k, for $k \in \{1, 2, ..., N\}$.

When building our training sets, we used simple random sampling. Where our methods did struggle, we believe this difficulty is due to the inability of a deterministic surrogate to successfully emulate the sensitivities of a model, rather than the choice of sampling method.

To compute the sloppy/stiff eigenvectors at a new point in parameter space, Θ^{new}, compute the surrogate Jacobian by differentiating the surrogate output $S(\Theta^{\mathbf{new}})$ numerically (or via backpropagation). This process yields Jacobian J. Compute the eigenvectors of matrix $J^T J$ to determine the sloppy/stiff directions.

As is true for all neural networks, the ability of a neural network surrogate to achieve its design goals depends on the volume and quality of training data, the chosen network structure (number of nodes, number of layers, type of layer, etc) and the training technique (including choice of loss function).

2.4.1 Parameter Normalisation

As discussed in Section 0.4.5, it is standard to use log-parameters when sloppy eigendirections are being computed, if the parameters are of different orders of magnitude. However, because we are making use of neural networks in this chapter, it is convenient to instead choose a parameter range for each raw parameter (say, $\theta_i \in [a_0, a_1]$), and use a dummy parameter $\hat{\theta}_i$ sitting in the range $[0, 1]$ that maps onto this range:

$$\theta_i = a_0 + \hat{\theta}_i \times (a_1 - a_0). \tag{2.2}$$

The dummy parameters $\hat{\theta}_1, \hat{\theta}_2, ...$ will be used as inputs to the neural network surrogate. The model outputs (X_t) will also be normalised, so the neural network only needs to produce values in $[0, 1]$.

2.5 Model Features to Emulate

Before we build a surrogate, we must first decide which of the model's outputs we wish to generate/emulate: the full model outputs, a set of summary statistics, or something else. For a multivariate time series model (e.g., a DSGE or an ABM), the full model output would be $f_v(\theta, t)$, for all variables (v) and time (t) values.

Alternatively, we could follow the example of other authors, and use lower-dimensional summaries of the full model's outputs. These could be the first few moments of the output, automatically generated statistical features, or other metrics resulting from different dimensionality reduction techniques.

For instance, let us consider the endogenous business cycle ABM of Asano et al. [10] (detailed below in section 2.6.5). Many possible metrics are possible, shown below in ascending order of aggregation:

1. For every agent, the values for capital, savings rate, consumption, for every point in time
   - $y_{n,v,t}$, for $n \in \{1, 2, ..., N\}$, $v \in \{k, s, c\}$, $t \in \{1, 2, ..., T\}$
   - Output has size $N \times 3 \times T$
2. Aggregate over agents
   - Three time series: K_t, S_t, and C_t
   - Output has size $3 \times T$
3. A single aggregated time series (e.g. total capital at each point in time)
   - i.e., one univariate time series, K_t
   - Output has size T
4. Summary statistics, aggregated over agents and time
   - e.g., the first M moments of the total capital time series
   - Output has size M

When we use multi-channel outputs (case 1 above with N channels, each with output length T), we concatenate these together to form a single time series of length $N \times T$. The ordering of concatenation does not matter (i.e. K,S,C is no better or worse than C,K,S), because a fully connected network does not initially assume there is anything notable about the relationship between one pair of input nodes over another pair. Such relationships – which could represent temporal dependencies – are learnt during training. However, if recurrent or convolutional neural networks were used, the positioning of each time series in the concatenation may be relevant.

This chapter treats some models as multi-channel and others as single-channel. For example, all four channels of the Hodgkin-Huxley model are used, as they each behave distinctly from one another. Conversely, only one channel of the Agent-based SIR model was used, as all channels follow a near-identical temporal pattern.

2.5.1 Neural Network Candidates

The primary outputs of the economic models we consider are in the form of time series (possibly multivariate). If dimension-reduction techniques are applied (e.g., considering the moments of the time series to be the model's output), a simple FNN is appropriate. If we instead wish to consider the whole time series without reducing its dimension, a recurrent neural network[4] seems natural. However, it is also possible to use a FNN over the full time series (with one output node per time series point), and our experiments found this method successful in the models we investigated, despite the simplicity of the FNN. A description of the FNN design follows.

FNN Network Architecture

Figure 2.2 illustrates the FNN surrogate architecture, together with a summary of the process used to produce a trained surrogate.

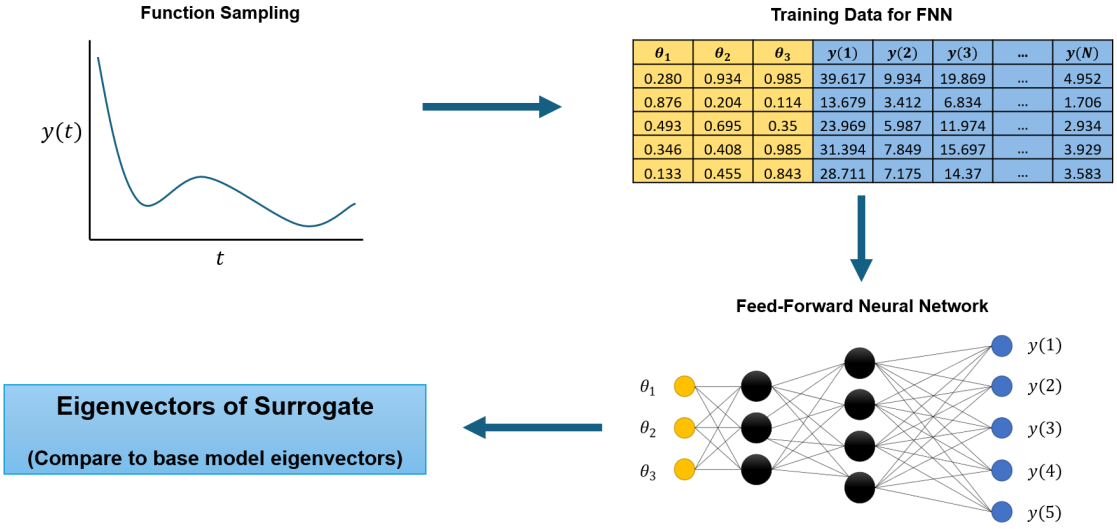


Figure 2.2: **FNN surrogate construction procedure**. For each parameter combination, the time series function is sampled at some or all points of its output, and this data is used to train the FNN. The FNN has one input node for each parameter, several hidden layers (only two shown), and one output node for each output time period. Eigenvectors can then be computed using numerical, analytical or automatic differentiation.

Constructing the FNN and Training Data

[4]This could make use of the structure of a simple RNN, long short-term memory (LSTM), gated recurrent units (GRU), or other structures.

Below is a description of how the FNN surrogates were constructed, including the structure of the networks' components. The precise structure of the networks used for each model can be found in Appendix C.2.

1. **Input Layer**: one node for each parameter.
2. **Hidden Layers**: five to ten densely-connected hidden layers were used.
3. **Loss function**: reconstruction error (L_2 norm) was used a loss function.
4. **Training set**: Around 1,000 - 2,000 runs of the target model were generated to serve as training and validation sets (80% for training, 20% for validation). Parameters vectors (in their normalised form) were drawn from the multivariate uniform distribution $[0, 1]^p$.

A diagram illustrating the full structure of the surrogate network used for the SIR model can be found in Figure C.1 in the appendix.

Training the Network

This step is the same as the one used in the creation of deterministic surrogates (Chapter 2). Specifically, the steps involves are:

1. Sample from the parameter space N times, producing parameter vectors $\Theta^1, \Theta^2, \ldots, \Theta^N$.
2. Run the model on those N parameter vectors to produce time series $G^1, G^2, \ldots, G^N$.
3. Train the surrogate, using pairs (G^k, Θ^k) as training examples.

2.6 Models

This chapter's goal is to develop a method applicable to economic models beyond DSGEs, with a focus on ABMs. We test our method on several mathematical models, as a proof of concept. The models we have chosen are:

- A simple toy function
- A differential equation model of biological neurons
- An ABM which models disease transmission
- An ABM built to model financial market ecology

- An ABM built to simulate emergent oscillations in simple economic systems
- An ABM representing a multi-sector macroeconomy

The four ABMs were selected as they have distinct structures, and stem from different areas of research within economics and epidemiology. This makes them useful in testing the applicability of our methods to different types of ABM. While the differential equation model of biological neurons is not an ABM, it is a suitable test case due to its nonlinear structure and diverse range of behaviours, which create challenges for surrogate modelling.

The parameter values, initial conditions, and SFD techniques used for each model are presented in Appendix C.1.

2.6.1 A Toy Function, $\phi(\theta, t)$

This simple four-parameter function serves as a basic test case for our surrogate method. The model contains additive Gaussian noise, and is defined by:

$$\phi(\theta, t) = \exp(-2\theta_0(t+1)) + \theta_0 + \theta_1 + \sin(\tfrac{1}{5}\theta_0) + \theta_2\theta_3 + \sin(\theta_2\theta_3) + \epsilon_t. \tag{2.3}$$

The function's structure and the constants (e.g. $\sin(\frac{1}{5}\theta_0)$) have been set in such a way that the stiff and sloppy eigendirections are not dominated by a single parameter.

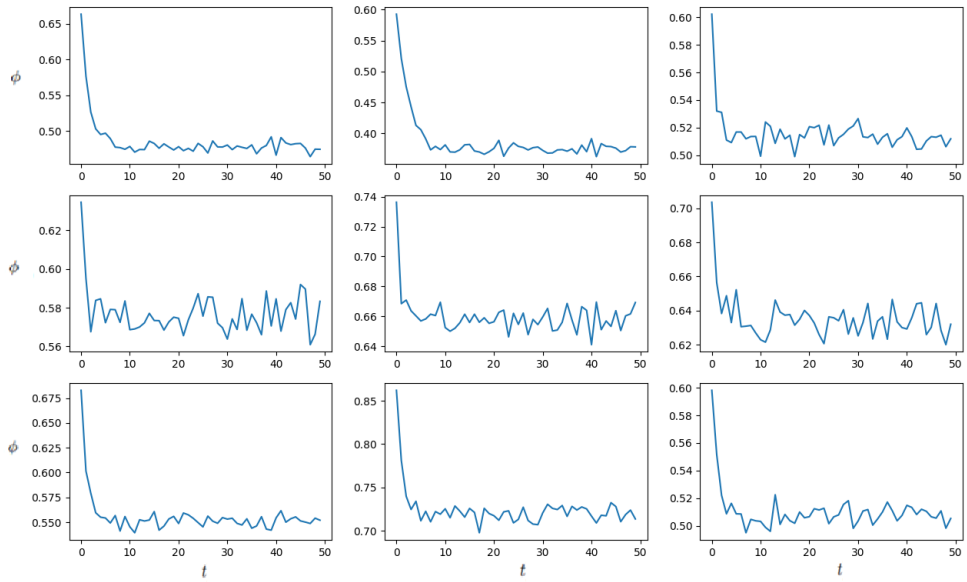


Figure 2.3: **Toy model $\phi(\theta, t)$ simulations**, each generated using different parameter values. Parameter ranges are listed in Appendix C.1.1.

2.6.2 Hodgkin-Huxley Model

Introduced in 1952, the Hodgkin-Huxley (HH) model seeks to simulate the generation and propagation of electrical potential energy in biological neurons [74]. Its creation led to its authors being awarded the Nobel Prize in Physiology or Medicine in 1963. The HH model consists of a set of differential equations with seven[5] parameters: the capacitance of the lipid bilayer (C_m), the maximum conductances ($\bar{g}_{\mathrm{K}}, \bar{g}_{\mathrm{L}}, \bar{g}_{\mathrm{Na}}$), and the source voltages ($E_{\mathrm{Na}}, E_{\mathrm{K}}, E_{\mathrm{L}}$). The time-dependent variables are the membrane voltage (V), and three gating variables (m, n, h).

The equations of the model are as follows. The *membrane equation* is:

$$\frac{dV}{dt} = \frac{-I_{\mathrm{Na}} - I_{\mathrm{K}} - I_{\mathrm{L}}}{C_m}. \tag{2.4}$$

The equations for the electric current variables are:

$$I_{\mathrm{Na}} = \overline{g}_{\mathrm{Na}} m^3 h (V - E_{\mathrm{Na}}) \tag{2.5}$$

$$I_{\mathrm{K}} = \overline{g}_{\mathrm{K}} n^4 (V - E_{\mathrm{K}}) \tag{2.6}$$

$$I_{\mathrm{L}} = \overline{g}_{\mathrm{L}} (V - E_{\mathrm{L}}), \tag{2.7}$$

where E_{Na}, E_{K}, and E_{L} are *source voltages*. The gating variable equation for each of $x \in \{m, n, h\}$ is:

$$\frac{dx}{dt} = \alpha_x(V)(1 - x) - \beta_x(V)x, \tag{2.8}$$

where $\alpha_x(V)$ and $\beta_x(V)$ are functions of the voltage, representing voltage-dependent rate constants for the ion channels. These functions are given by:

$$\alpha_n(V) = 0.01(V + 55)/(1 - e^{-(V+55)/10)}) \tag{2.9}$$

$$\beta_n(V) = 0.125e^{-(V+65)/80} \tag{2.10}$$

$$\alpha_m(V) = 0.1(V + 40)/(1 - e^{-(V+40)/10)}) \tag{2.11}$$

$$\beta_m(V) = 4.0e^{-(V+65)/18} \tag{2.12}$$

$$\alpha_h(V) = 0.07e^{-(V+65)/20} \tag{2.13}$$

$$\beta_h(V) = 1/(1 + e^{-(V+35)/10}). \tag{2.14}$$

[5]Some versions of the HH model include I_{ext} (an external current applied to the membrane) as an additional changeable parameter, or exogenous, time-dependent variable [66]. Alternatively, this can be set to zero when considering the behaviour of a neuron without external stimulation [2]. We will use a fixed value of $I_{\mathrm{ext}} = 10$ in our analysis.

While the HH model is not an economic model, it has been chosen as a test case due to its complex nonlinear behaviour [66, 53, 54, 95] (as shown in Figure 2.4), and its use in other studies as a test case for different numerical methods [29, 154, 14, 60]. As we wish to test our surrogate method on stochastic models, the HH model used in this chapter has stochasticity in the form of additive Gaussian noise attached to each time-dependent variable ($V(t)$, $m(t)$, $n(t)$, $h(t)$) at each timestep.

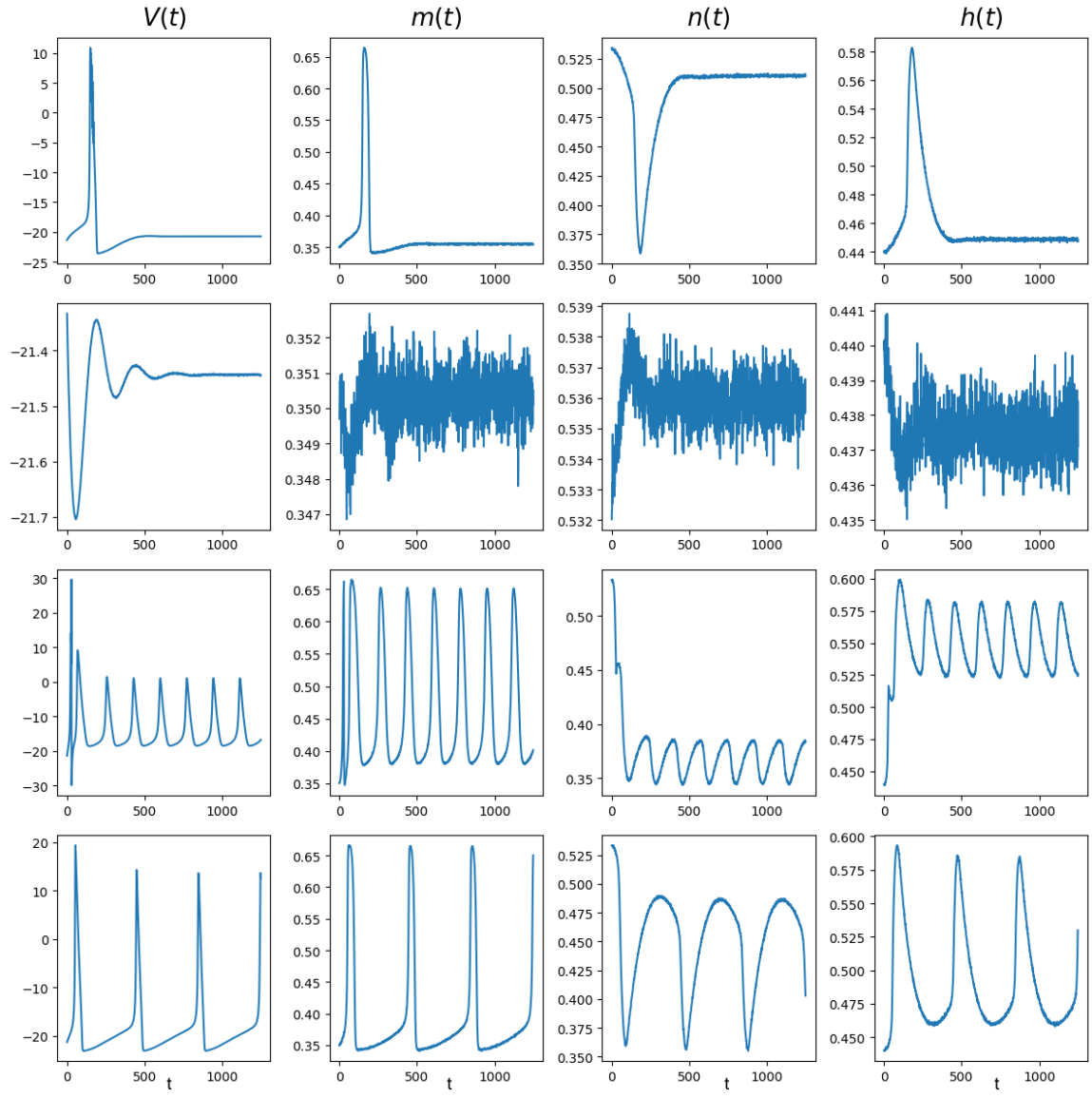


Figure 2.4: **HH simulations.** Four realisations of the HH time series model, illustrating the dynamics of the four channels of the multivariate output time series (V, m, n, h). Varying the parameters can result in significantly different behaviours, demonstrating the HH model's complex response patterns. Parameter ranges are listed in Appendix C.1.2.

2.6.3 An Agent-based SIR Model

This is a model of interacting agents who can spread an infection to each other. It takes the underlying assumptions of a standard SIR (susceptible, infectious, recovered) model, but instead of using differential equations to simulate rates of contact and recovery, it simulates individual people randomly interacting and/or recovering at each timestep[6].

In this model, a "neighbourhood" of individuals is created, represented by a Watts-Strogatz (WS) network. The neighbourhood represents individuals' possible contacts, governing the transmission of the disease.

Background: Watts-Strogatz Networks

Introduced in 1998, WS networks are graphs that can represent connections between individuals. Notably, WS networks show relatively high clustering – meaning if node A is directly connected to nodes B and C, there is a high probability B and C are also connected to one another by a single edge. This feature is not seen in Erdős–Rényi networks, where clustering is comparatively rare, making WS networks more suitable to study some social phenomena.

A WS network can be constructed using three parameters (N, K, β), using the method described below.

1. N nodes are arranged in a ring lattice, with each node having two neighbouring nodes.

2. K (an even integer) represents the degree of each node. For each node, edges are added until it is connected to the closest $K/2$ nodes in the clockwise direction around the ring, and the closest $K/2$ nodes in the anticlockwise direction.

3. Rewiring: For each node j, consider the edges that connect it to its $K/2$ clockwise neighbours, $E_{j,cw}$. For each edge in $E_{j,cw}$, leave it in place with probability $(1-\beta)$ rewire it to a random node in the network with probability β, and leave it in place with probability $1-\beta$.

An illustration of these steps involved in graph creation (initialising the edges with nearest-neighbours, and random rewiring) is shown in Figure 2.5.

[6]The code from which our implementation is based upon can be found at [82].

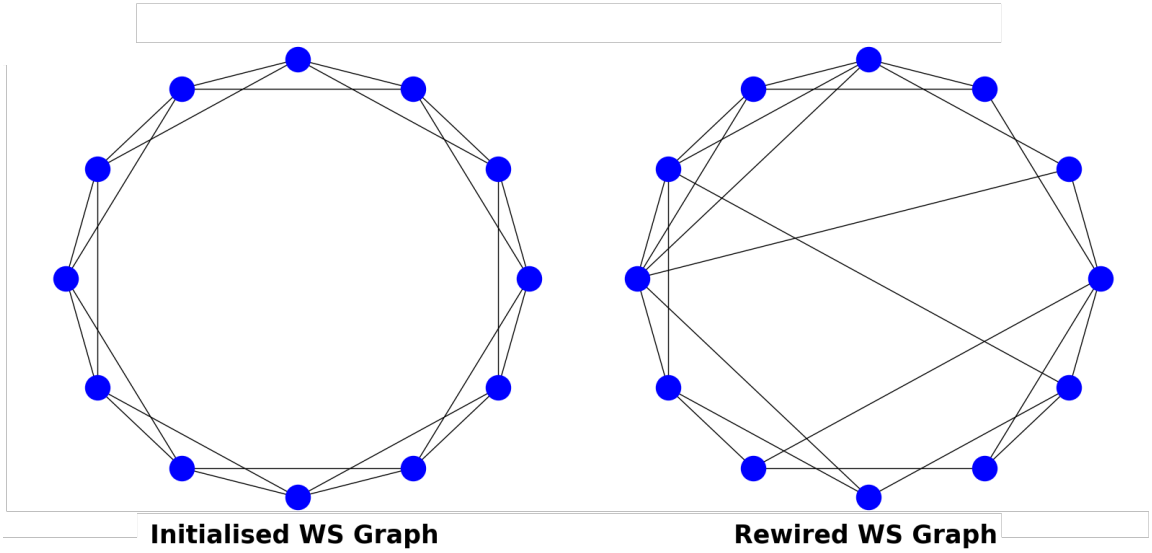


Figure 2.5: **Construction of a WS network**. The steps shown are: (1) initialisation (nodes attached to nearest neighbours), and (2) random rewiring. (Parameters used for constructing this network: $N = 12$, $K = 4$, $\beta = 0.2$.)

SIR Model Rules

Each individual can be in one of three different states: susceptible (S), infected (I), or recovered (R). The spread of infection is determined by interaction rules. Initially, a small proportion of the population is infectious, and all others are susceptible. In each time period,

- If the jth individual (H_j) is in state I, for each susceptible person in the neighbourhood of H_j, there is a probability p_I that they are infected by H_j.
- Each infected person has a probability p_R of recovering, in which case their state changes from I to R.

The model parameters we vary in our analysis are infection probability (p_I), recovery probability (p_R), initial infection proportion of the population (λ_0), and the rewiring probability of the WS network (β).

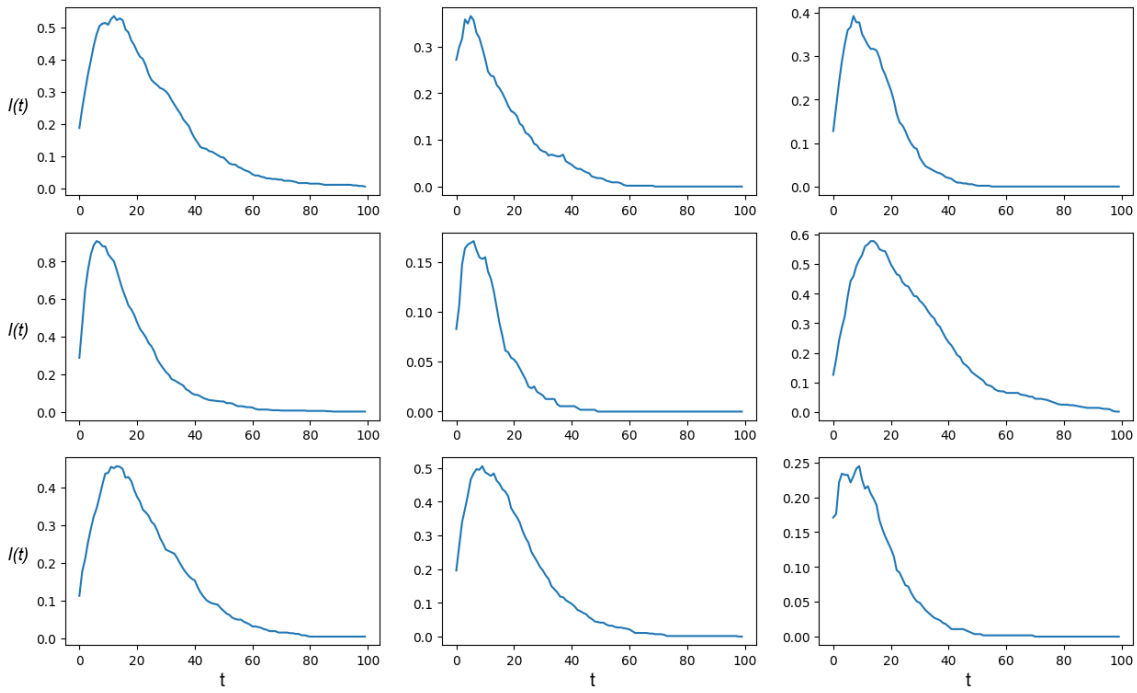


Figure 2.6: **Agent-based SIR simulations**, each generated using different parameter values. The dependent variable is the population infection rate. Parameter ranges are listed in Appendix C.1.3.

2.6.4 Brock-Hommes Model

The Brock-Hommes (BH) ABM consists of heterogeneous agents interacting in a financial ecosystem. The BH model has been used to study complex, emergent behaviours in financial markets [26].

The rules of the BH model are as follows. At each timestep:

1. The price (P_t) of an asset is updated according to the previous price (P_{t-1}) and the aggregate excess demand (D_{t-1}) for the asset
    - $P_t = P_{t-1} + \lambda D_{t-1}$.
2. Agents follow one of multiple strategies, and may switch to an alternative strategy if it is presently outperforming their own. Such strategies include the following:
    - "Chartist" strategies are founded on a belief that recent price trends help predict future prices.
        - Price expectations follow autoregressive rules, such as $\mathbb{E}_h[P_t] = a_h + b_{h,1}P_{t-1} + b_{h,2}P_{t-2}$.

- Chartist strategies may be momentum-based (where price trends are expected to continue), or mean-reverting, depending on the values of $a_h, b_{h,1}, b_{h,2}$.

- "Fundamental" pricing strategies entail agents estimating the "true" value of an asset, $\mathbb{E}[P_t] = P^f$, independent of the current price and trend.

3. Demand for the asset from agent i is a function of the current price, and agent i's expected value for the future price.

   - $d_i(t) = \alpha \times (\mathbb{E}_i[P_t] - P_{t-1})$
   - Excess demand is $D(t) = \Sigma_i d_i(t)$

The BH variant used is the same as that used by Platt [117]. The four parameters we vary in our experiments are expectation formation parameters (a_1, b_1, a_2, b_2) for two chartist strategies.

Simulations of the BH model are shown in Figure 2.7.

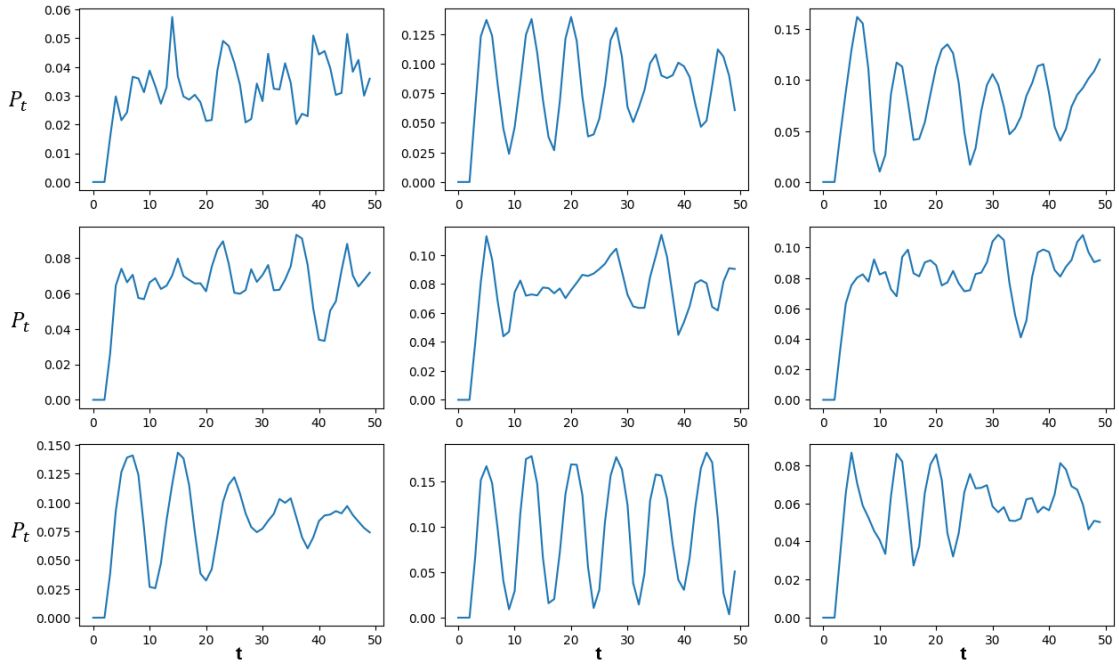


Figure 2.7: **Brock-Hommes simulations**, each generated using different parameter values. The dependent variable is the asset price, P_t. Parameter ranges are listed in Appendix C.1.4.

2.6.5 An Agent-Based Ramsey–Cass–Koopmans Model

The model created by Asano et al., 2019 [10] is an agent-based version of the Ramsey–Cass–Koopmans (RCK) model – a neoclassical model of economic growth. The agents of the model are households, which each possess capital, work to earn wages, and consume goods based on their total earnings.

In this model, households exist within a "neighbourhood" (represented by an Erdős–Rényi network, or a fully connected network). Households earn income through wages and investment returns on capital they own. They can split this income between consumption and savings (which builds a household's capital), according to their savings rate (s_i). Households are free to choose their own savings rate, and use a heuristic to do so: at random time periods (determined by a Poisson process), household H_i will observe all the households in its neighbourhood. If any neighbours are consuming more than this household at the current time step, it will imitate the most "successful" neighbour by copying its savings rate (s_j). This mimicry may or may not lead to a better outcome for that household: copying a neighbour who has been consuming rather than saving (i.e., small s_j) will lead to greater short term consumption, but at the cost of future earnings and consumption. This behaviour may later lead to the household experiencing falling consumption, at which point they may imitate a high-savings household which is consuming more than them.

With this "naive imitation" form of bounded rationality, aperiodic oscillations can form endogenously – often due to agents rapidly switching between strategies because increasing in their saving rate does not reap immediate benefit. This is especially the case when the time between updates (τ) is very short. These oscillations occur at the level of individual households, and at the level of macroeconomic aggregates (e.g., total output, total capital). They are analogous to business cycles observed in the real economy, in which output and consumption vary over time. This model demonstrates a way in which business cycles can emerge without external shocks (in contrast to DSGE models, which can only generate oscillations when exogenous shocks are present).

The model possesses a set of parameters, which includes investment return per unit capital (α), capital depreciation rate (d), standard deviation of initial household labour endowment[7] (σ), and mean interaction time for the Poisson process (τ).

[7]Each household is endowed with a different quantity of labour, meaning households earn different wages from one another.

As output from the ABM, we take the time series of the total income of all agents over 1500 time steps, and exclude the first 500 periods to remove an erratic transient period occurring at the beginning of the simulations.

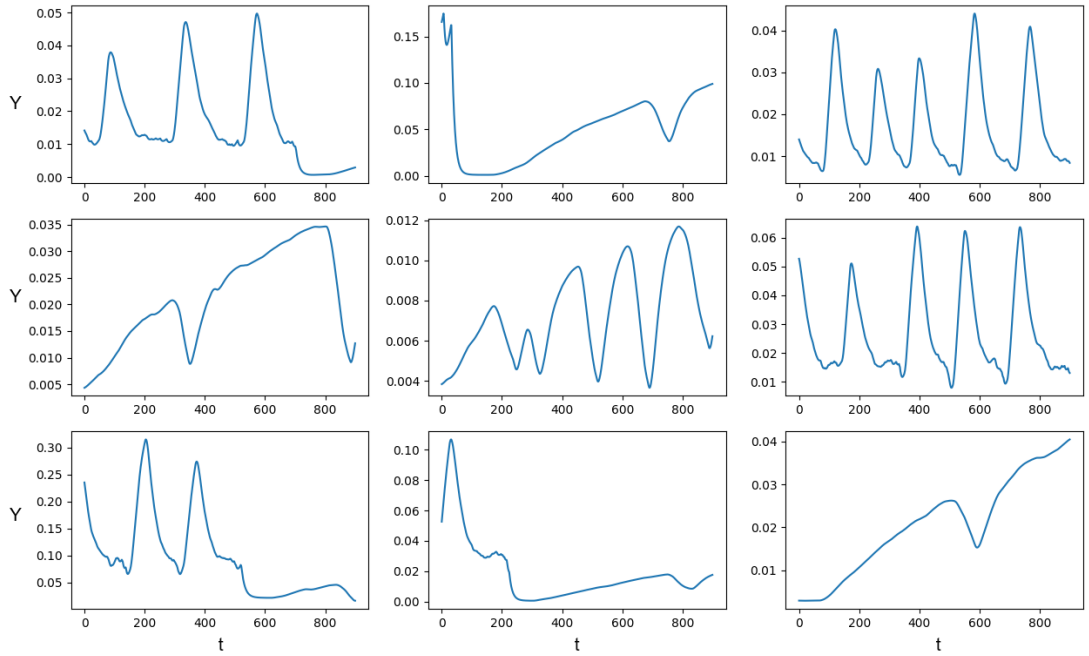


Figure 2.8: **Agent-based RCK simulations**, each generated using different parameter values. The dependent variable is total economy-wide income. Parameter ranges are listed in Appendix C.1.5.

2.6.6 Mark-0 Agent-Based Model

The Mark-0 ABM is a multi-sector economic model comprised of workers, firms, and a banking sector [65]. It possesses a large and varied phase space, and can produce a range of emergent phenomena. The phenomena the model is capable of exhibiting include stable equilibria, labour market oscillations, and self-perpetuating unemployment crises. It has been used to analyse labour market dynamics, and study the economic impact of the COVID-19 pandemic [130], including the effect of lockdowns and the subsequent economic recovery. This model makes a useful test case for our methods, as it is an example of an ABM of significant scale that has seen real-world applications. Additionally, it has a large set of changeable parameters – of which we use 10 – which makes it a good test of our surrogate method's ability to handle many-parameter models.

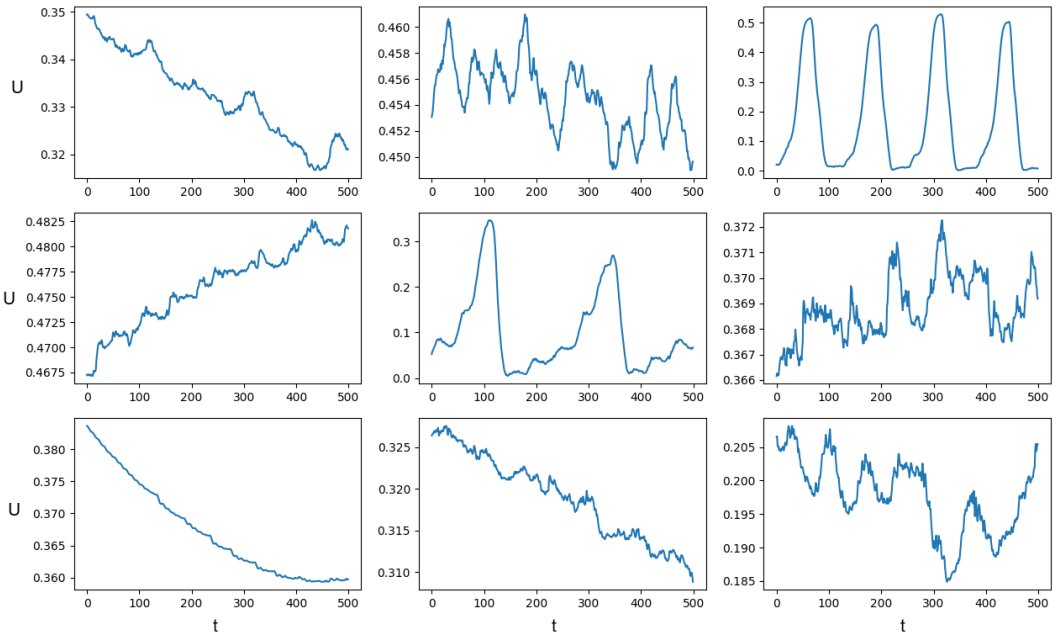


Figure 2.9: **Mark-0 simulations**, each generated using different parameter values. The dependent variable is the rate of unemployment. Parameter ranges are listed in Appendix C.1.6.

2.7 Results

As in Chapter 1, several methods were used to evaluate the ability of our surrogation technique to identify the stiff and sloppy eigendirections of the models of interest: the inspection of the eigenvalue spectra, CS matrices, and RMSE values. In this section, we first go through each model individually, and display (a) a plot illustrating the similarity between the model and its own surrogate (with the surrogate's output plotted alongside the model's own output), and (b) CS matrices to compare the SFD and surrogate eigenvectors. Subsequently, all the models' eigenspectra are shown on a single plot, along with a table of the RMSE values for all models. For each model, CS matrices in this chapter and the next were computed at 15 points in parameter space, and averaged to produce the matrices shown in the respective figures. The structures of the surrogates used for each model are detailed in Appendix C.2.

2.7.1 Toy Model $\phi(\theta, t)$ Results

Figure 2.10 shows the output of the deterministic surrogate for $\phi(\theta, t)$, along with several runs from the model itself. Figure 2.11a shows the CS between the eigenvectors

of the surrogate and those derived from SFD. The surrogate was able to identify the stiff/sloppy eigendirections accurately, although the sloppiest two eigenvectors were switched around. This switch might be explained by the third and fourth eigenvalues being so small they could not be accurately computed.

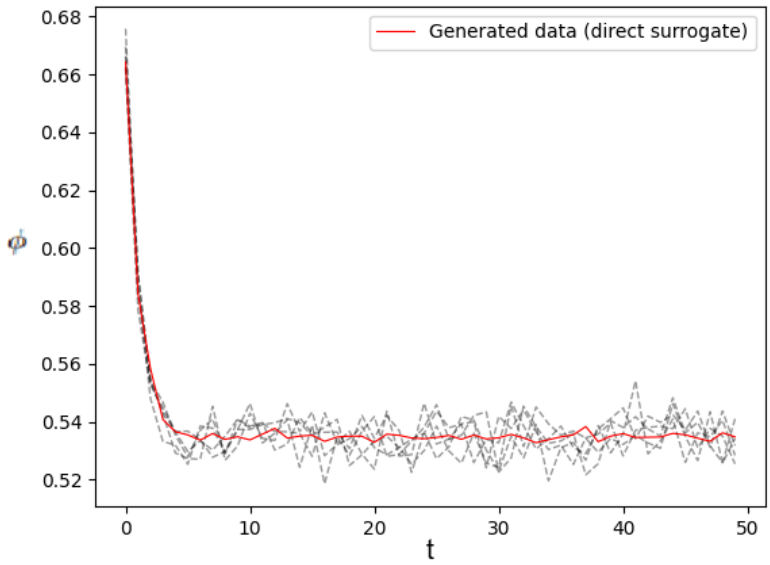
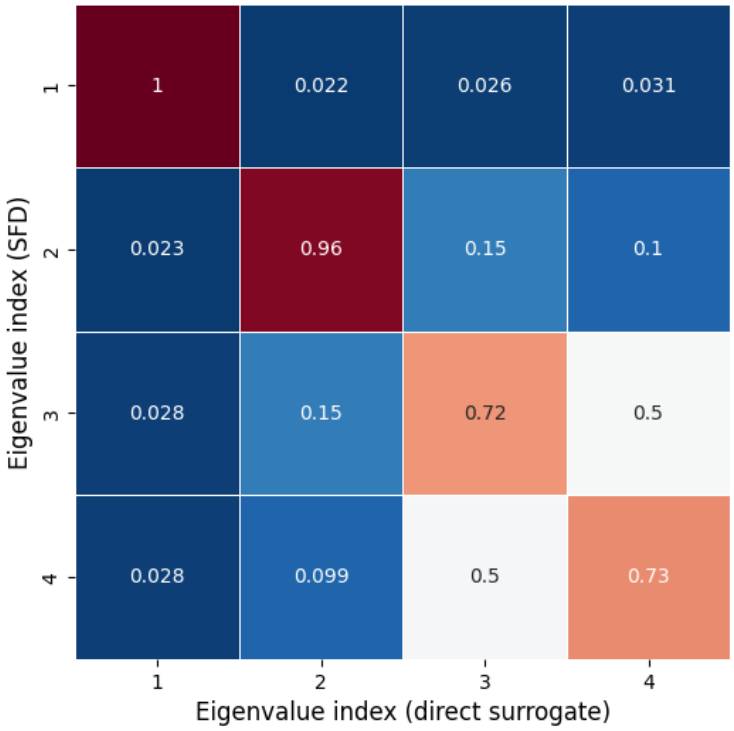
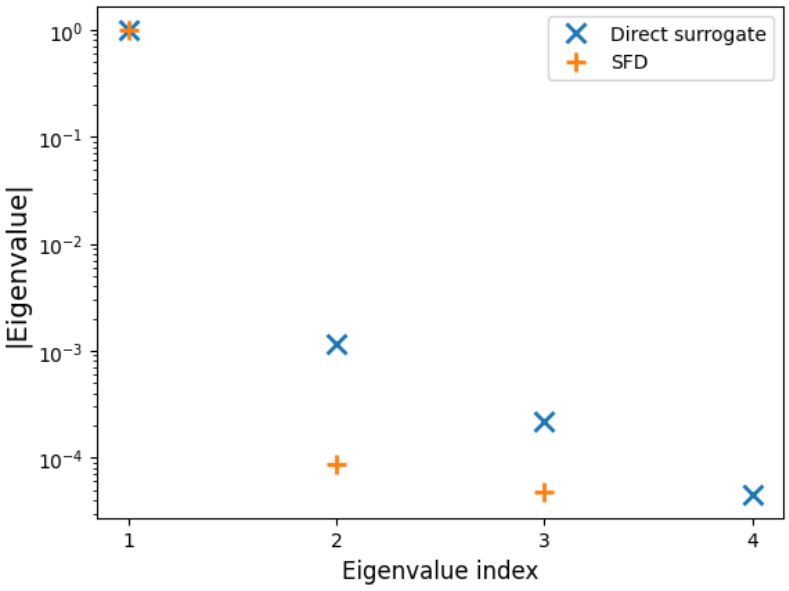


Figure 2.10: **Toy model $\phi(\theta, t)$ surrogate behaviour**. Deterministic surrogate output shown, along with several runs of $\phi(\theta, t)$ itself. The deterministic surrogate has learned to produce curves that approximately match the average of the oscillations at each point in time. Parameters were drawn from distributions specified in Appendix C.1.1.



(a) $\phi(\theta, t)$ cosine similarity matrix.



(b) $\phi(\theta, t)$ eigenvalue spectrum.

Figure 2.11: **Toy Model $\phi(\theta, t)$ surrogate analysis.** (a) Cosine similarity matrix: Strong alignment was observed for the first two eigenvectors, while some eigenvector swapping took place amongst the third and fourth. (b) Eigenvalue spectrum. The smallest SFD eigenvalue is not shown, as it was too small to be plotted with the other points. Fifteen different parameter vectors were used, each drawn from the distribution specified in Appendix C.1.1.

2.7.2 HH Results

Figure 2.12 illustrates the output of a trained FNN surrogate, along with the output of the HH model itself. While the surrogate did not always produce identical-looking time series, it generally produced time series that were qualitatively similar to the HH model's own outputs. Qualitative mimicking behaviour is a positive sign (as per Figure 2.12), but the primary goal of our surrogation method is to find the stiff/sloppy eigendirections. After choosing 15 points in parameter space, Jacobians were computed separately via SFD and backpropagation (using the surrogate) at each point. These Jacobians were then averaged, allowing us to compute FIMs and their eigenvectors. The degree of alignment between the SFD and surrogate eigenvectors is illustrated in a CS plot (Figure 2.13a).

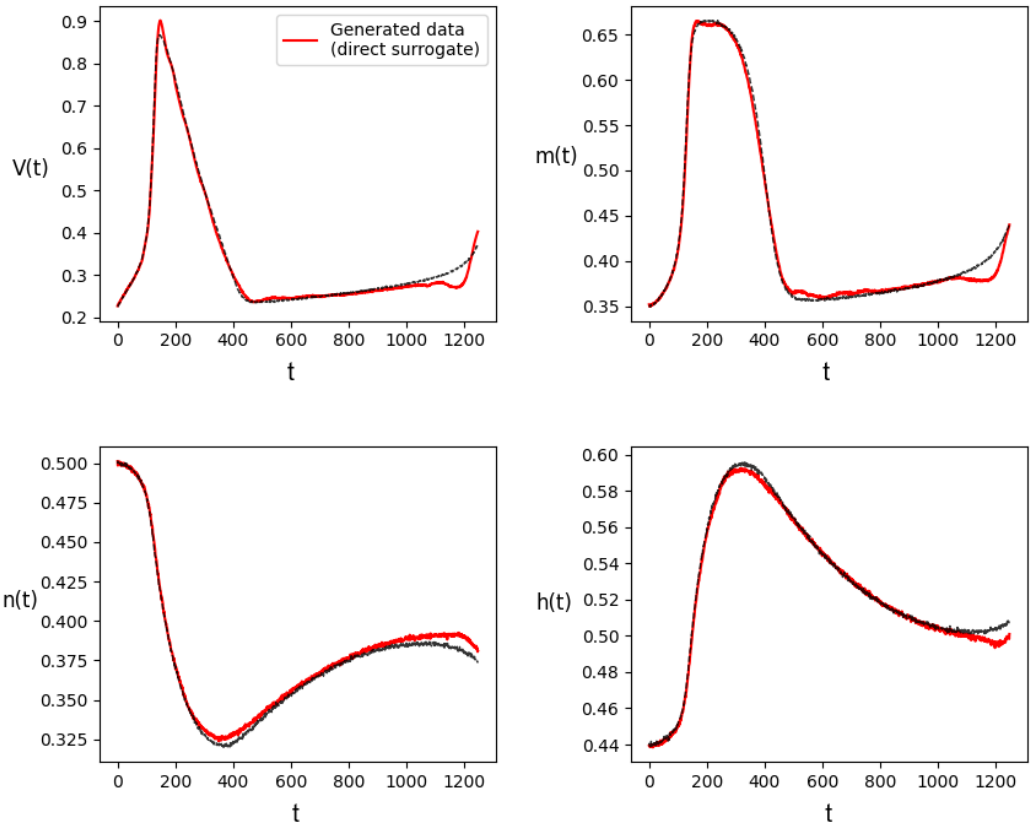
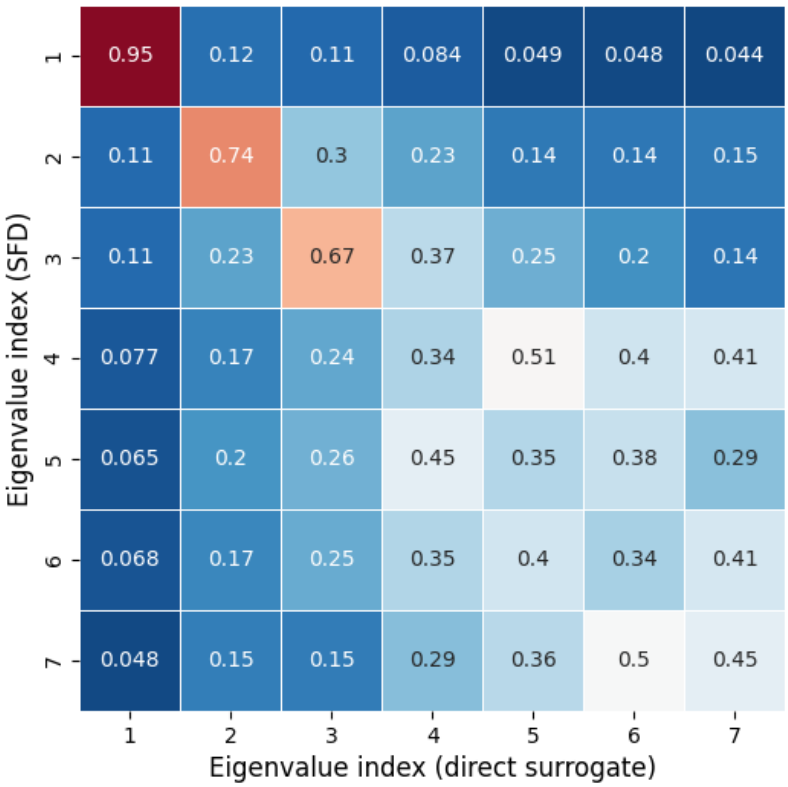
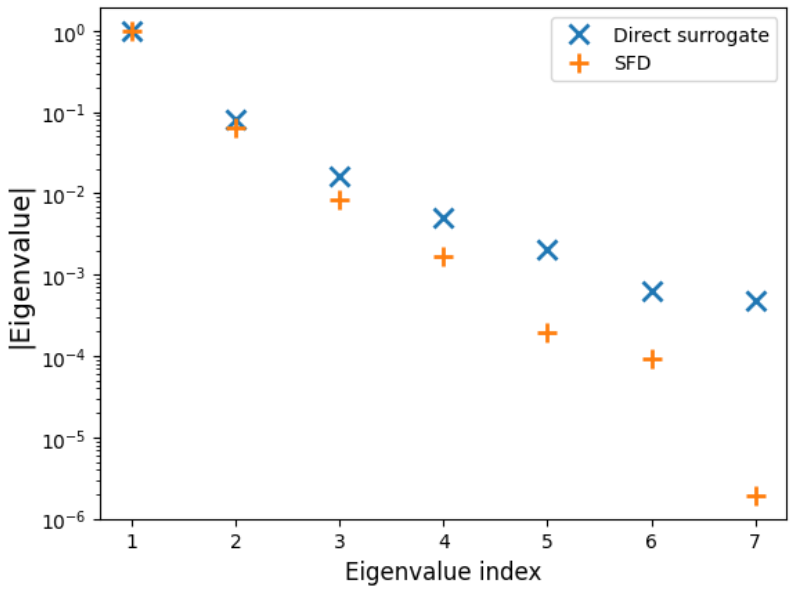


Figure 2.12: **HH surrogate behaviour**. Outputs from the HH model are plotted against data generated from the surrogate, using the same parameter values. Variables plotted are V, m, n, and h. Parameters were drawn from the distribution specified in Appendix C.1.2.



(a) HH cosine similarity matrix.



(b) HH eigenvalue spectrum.

Figure 2.13: **HH surrogate analysis.** (a) Cosine similarity matrix: Good alignment was observed for the first three eigenvectors. (b) Eigenvalue spectrum: FNN and SFD eigenvalues diverge significantly for sloppier eigenvectors. The model and surrogate were run on 15 different parameter vectors, each drawn from the distribution specified in Appendix C.1.2.

2.7.3 Agent-based SIR Results

The trained surrogate performed well in mimicking the SIR ABM's outputs, as shown in Figure 2.14. It was also able to accurately predict all of the sloppy and stiff eigenvectors of the model. Despite the non-Gaussian stochasticity of this ABM, the surrogate performs well.

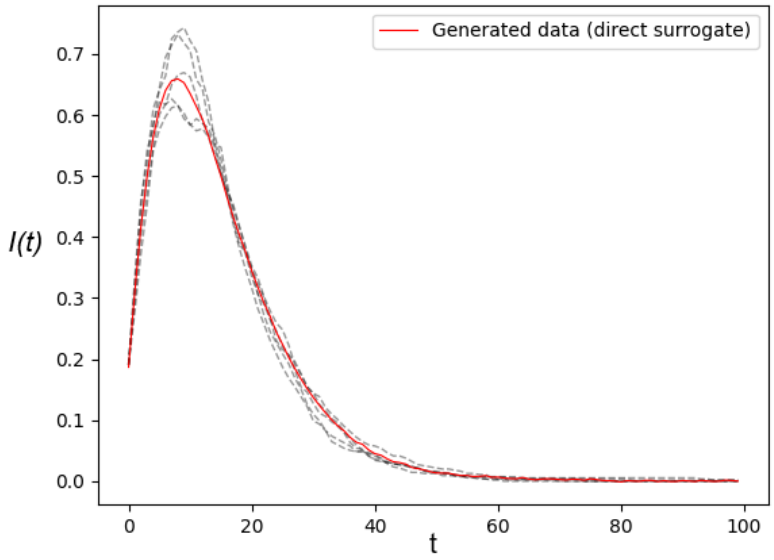
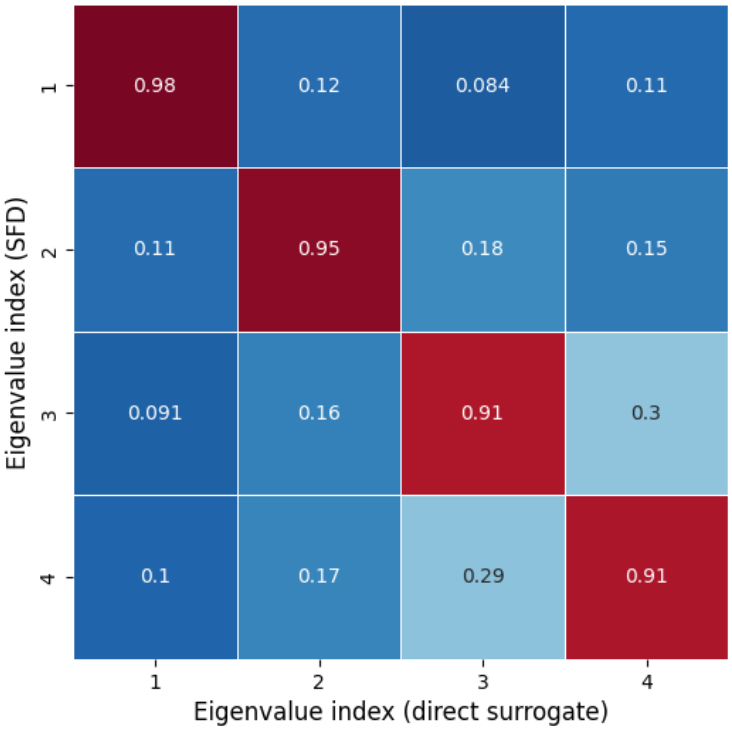
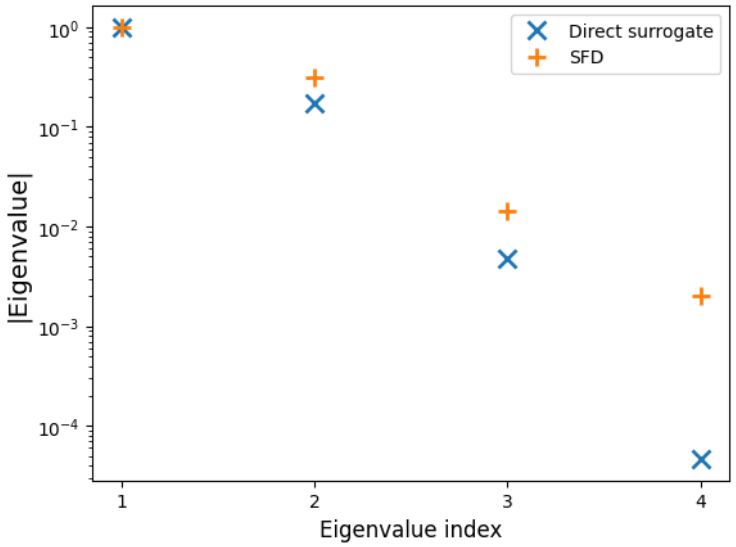


Figure 2.14: **Agent-based SIR surrogate behaviour**. Deterministic surrogate output shown, along with several runs of the SIR model itself. Parameters were drawn from the distribution specified in Appendix C.1.3.



(a) Agent-based SIR model cosine similarity matrix.



(b) Agent-based SIR model eigenvalue spectrum.

Figure 2.15: **Agent-based SIR surrogate analysis.** (a) Cosine similarity matrix: A high degree of alignment between all SFD-surrogate eigenvector pairs was observed. (b) Eigenvalue spectrum: The surrogate eigenvalue spectrum is relatively similar to the SFD eigenspectrum, which is unsurprising given the strong alignment observed in the cosine similarity matrix. Fifteen different parameter vectors were used, each drawn from the distribution specified in Appendix C.1.3.

2.7.4 BH Results

In the case of the BH model, the output of the surrogate sometimes closely matched that of the model, but at other times it was starkly different (see figure 2.16). It seems likely this is a result of the high degree of variability in the outputs of the BH model, even when parameters are varied minimally. Notably, the surrogate tended to perform best at predicting the first few values of the BH model, corresponding to the period of growth from the initial market price towards an equilibrium value, around which oscillations occur due to over- or undershooting. The subsequent oscillatory period was more difficult for the surrogate to mimic, likely as a result of the initiation of the oscillatory period being inconsistent between runs, even when θ was held constant.

As a result of the surrogate struggling to reliably emulate the BH model, the alignment between SFD and surrogate eigenvectors was weaker than in the case of the SIR ABM. Nevertheless, the stiffest eigenvector could be identified relatively reliably. The relative ease of identifying the stiffest eigenvector compared to the sloppiest was observed across multiple models – a phenomenon discussed further in Section 2.8.2.

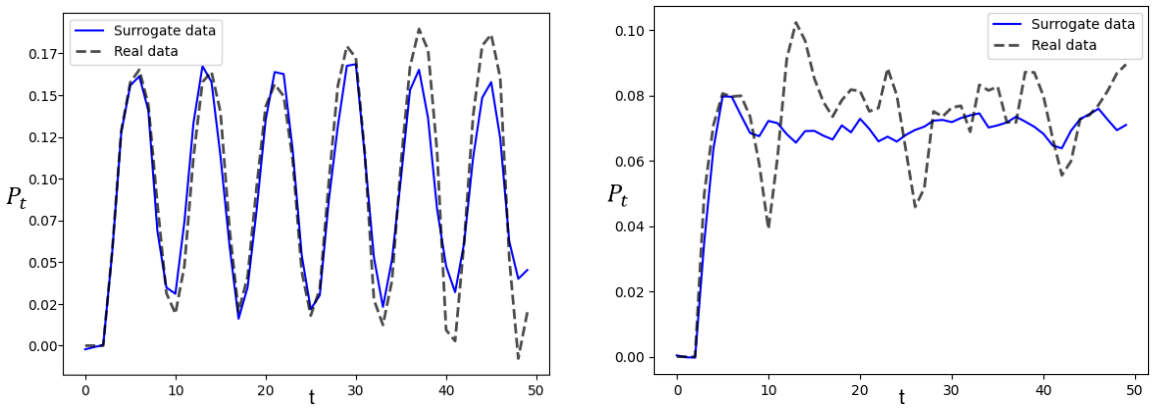
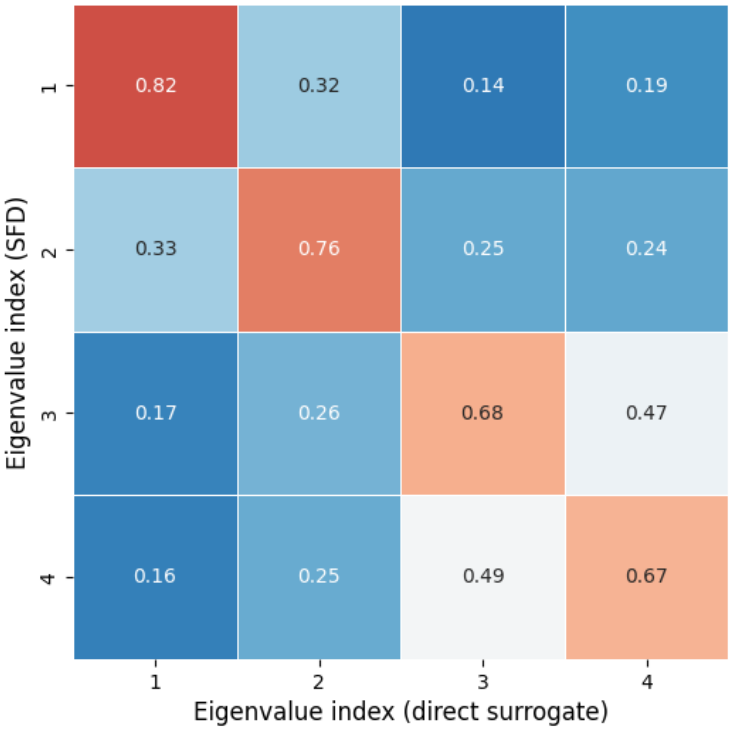
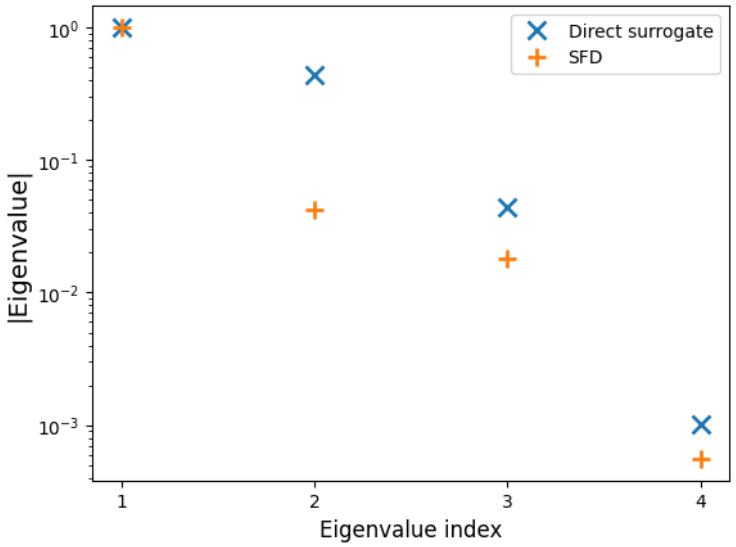


Figure 2.16: **BH surrogate behaviour**. Shown are simulations of the BH model at two points in parameter space, along with the deterministic surrogate's output with the same parameters. In some cases, the surrogate's output closely matches that of the model (as illustrated in the left plot), but other times the surrogate time series has a very different shape from that of the model. The two parameter vectors used were drawn from the distribution specified in Appendix C.1.4.



(a) Brock-Hommes cosine similarity matrix.



(b) BH eigenvalue spectrum.

Figure 2.17: **BH surrogate analysis.** (a) Cosine similarity matrix: The stiffest eigendirections of the SFD and surrogate methods are well-aligned, but less so for sloppier eigendirections. (b) Eigenvalue spectrum. Fifteen different parameter vectors were used, each drawn from the distribution specified in Appendix C.1.4.

2.7.5 Agent-based RCK Results

The surrogate method had significantly more difficulty identifying the stiff and sloppy eigendirections of the RCK ABM compared to the other models. This difficulty is illustrated in Figure 2.19a, which shows very poor alignment between SFD and surrogate eigendirections.

As was shown in figure 2.8, the RCK ABM output can vary greatly, even when the model is run with the same parameters. In particular, the amplitude, frequency and phase of oscillations can change noticeably each time the model is re-run. These features are consistent with the goal of the creators of this model – namely, to create a system demonstrating the emergence of endogenous business cycles – but these features do create challenges when trying to recover the stiff/sloppy eigendirections of the model. However, as we shall show in Chapter 3, it is possible to create a generative surrogate to recover some information on the stiff directions of this model. This ability implies a clear advantage for the generative surrogate method when the model of interest has a highly stochastic time series output.

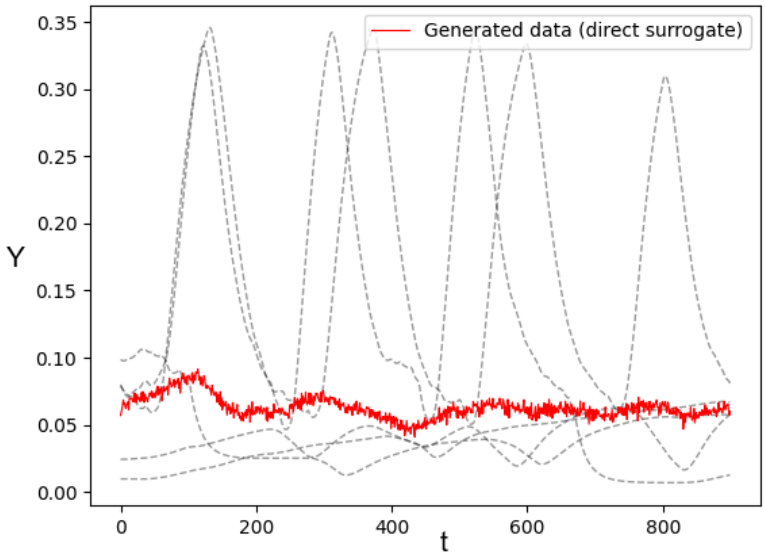
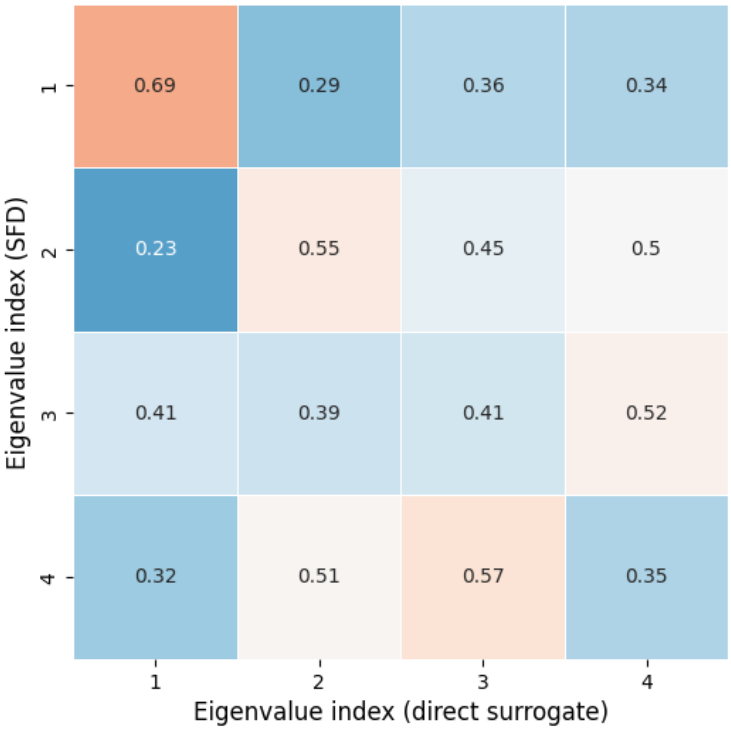
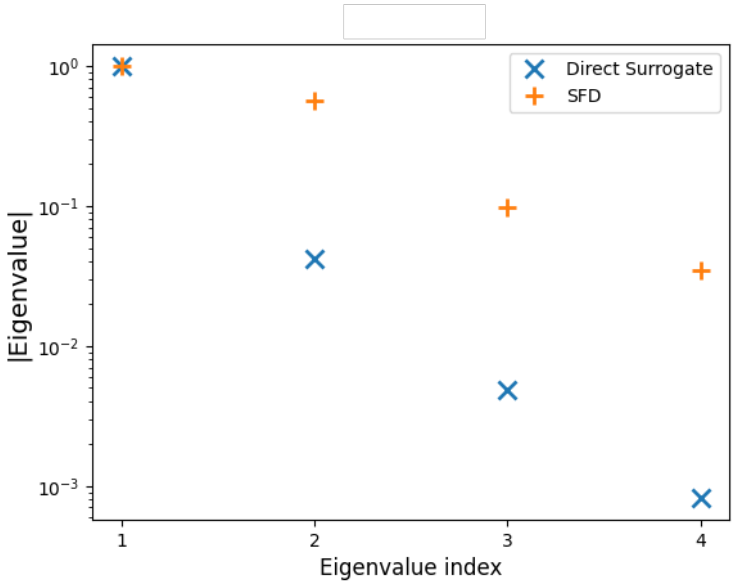


Figure 2.18: **RCK ABM surrogate behaviour**. Deterministic surrogate output shown, along with several runs of the RCK ABM itself. The deterministic surrogate does not replicate the large oscillations of the RCK ABM. Parameters were drawn from the distribution specified in Appendix C.1.5.



(a) RCK ABM cosine similarity matrix.



(b) RCK ABM eigenvalue spectrum.

Figure 2.19: **RCK ABM surrogate analysis.** (a) Cosine similarity matrix: Relatively little alignment is seen on the main diagonal, except for the stiffest eigenvector, indicating that the deterministic surrogate method has struggled to identify the stiff/sloppy eigendirections of the ABM. (b) Eigenvalue spectrum: Large divergences are visible, consistent with the surrogate's difficulty in predicting the eigendirections of the model. Fifteen different parameter vectors were used, each drawn from the distribution specified in Appendix C.1.5.

2.7.6 Mark-0 Results

Similar to the RCK model, the surrogate method had significantly more difficulty identifying the stiff and sloppy eigendirections of the RCK ABM compared to the other models. This challenge is illustrated in Figure 2.19a, which shows very poor alignment between SFD and surrogate eigendirections.

As is the case for the RCK model, in Chapter 3 we find that some eigenvectors can be accurately identified using generative surrogates.

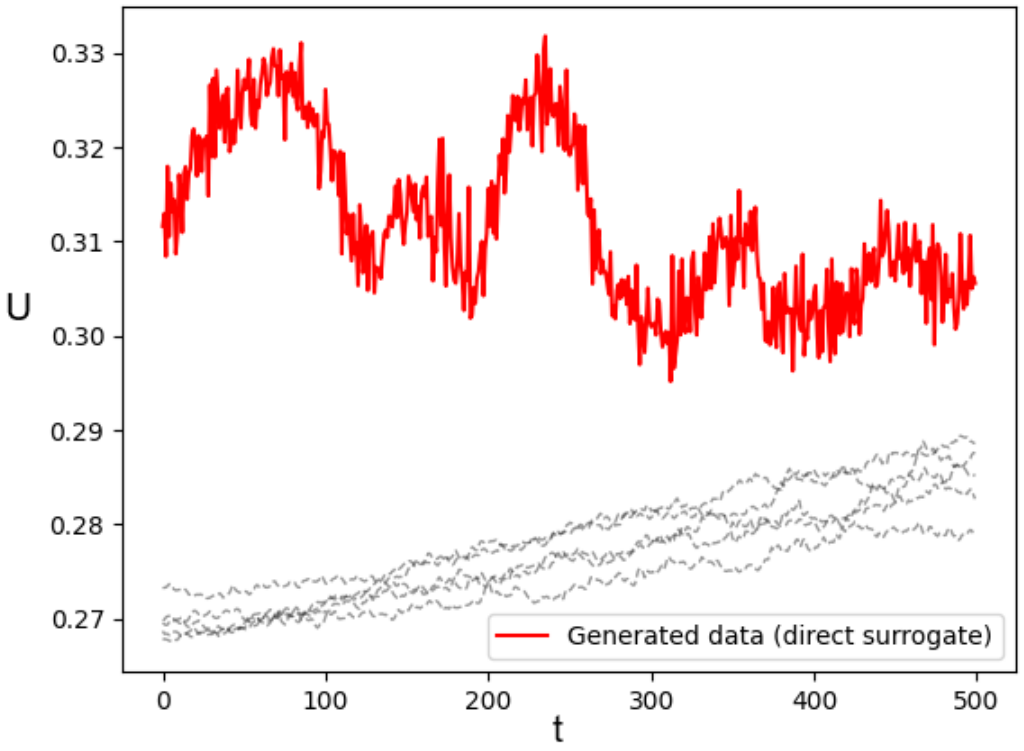
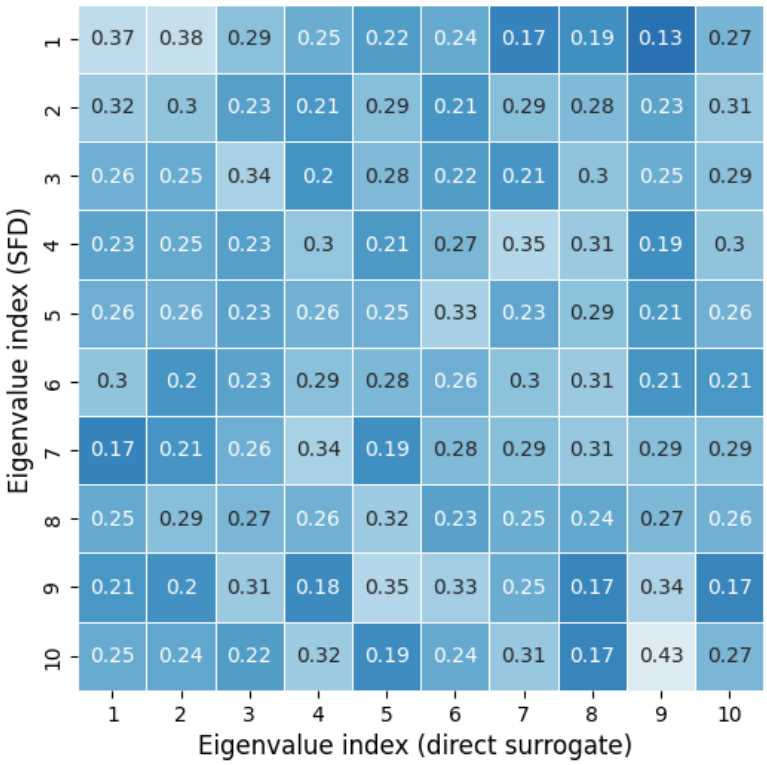
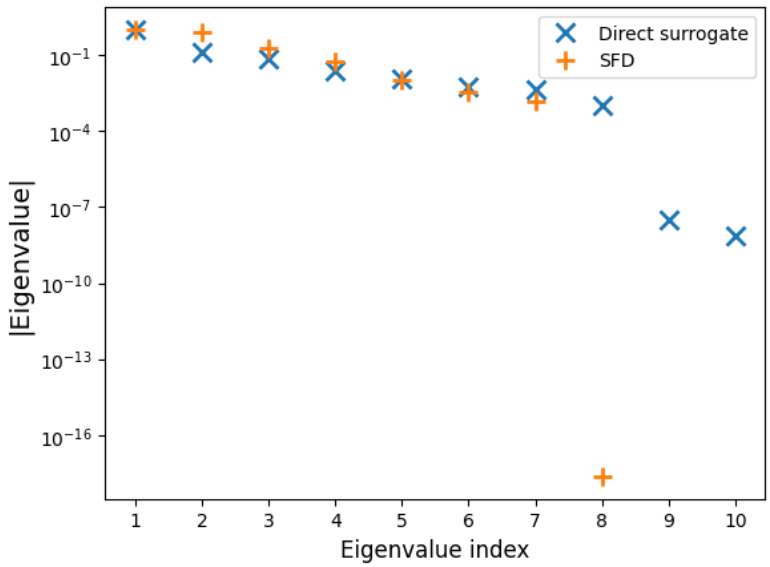


Figure 2.20: **Samples of the Mark-0**, along with the deterministic surrogate's output with the same parameters. The surrogate is unable to mimic the shape or magnitude of the Mark-0 time series. Parameters were drawn from the distribution specified in Appendix C.1.6.



(a) Mark-0 cosine similarity matrix.



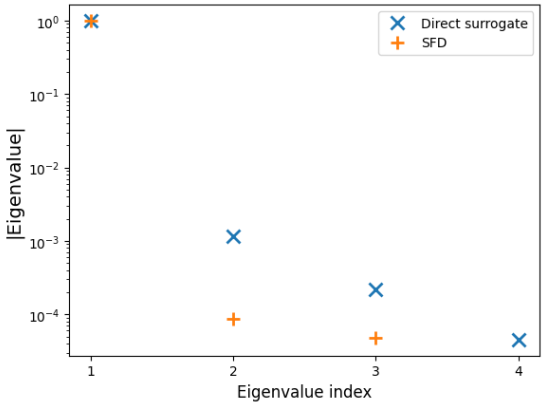
(b) Mark-0 eigenvalue spectrum.

Figure 2.21: **Mark-0 Surrogate Analysis.** (a) Cosine similarity matrix: Very little alignment seen for any eigenvectors, showing the deterministic surrogate has failed to identify stiff and sloppy eigendirections. (b) Eigenvalue spectrum: Large divergences are visible. Fifteen different parameter vectors were used, each drawn from the distribution specified in Appendix C.1.6.

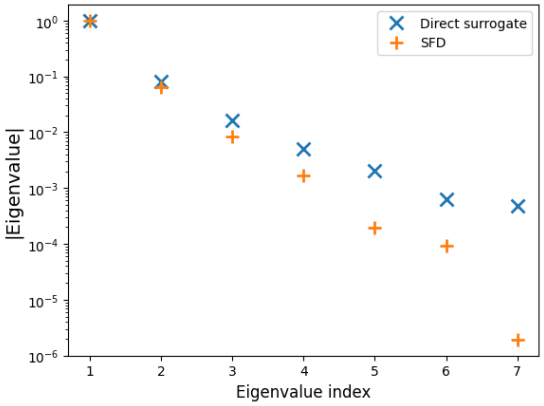
2.7.7 Eigenvalue Spectra – All Models

Plots showing the eigenvalue spectra for each model are shown in Figure 2.22. The eigenvalue spectra from the surrogate method differed noticeably from the SFD-derived spectra (with the exception of the Agent-based SIR model).

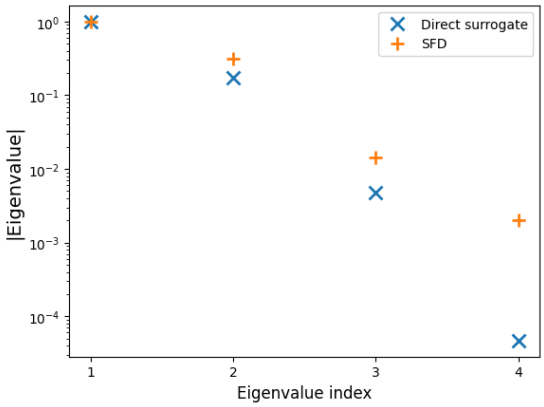
For all models, we found the surrogate's eigenvalue spectrum differs significantly from that of the respective model. In some cases this is substantial, for example the Brock-Hommes surrogate spectrum differed greatly from that of the model. This indicates there is a significant difference in the degree of parameter sensitivity between the surrogate and the model, but the reason is not clear.



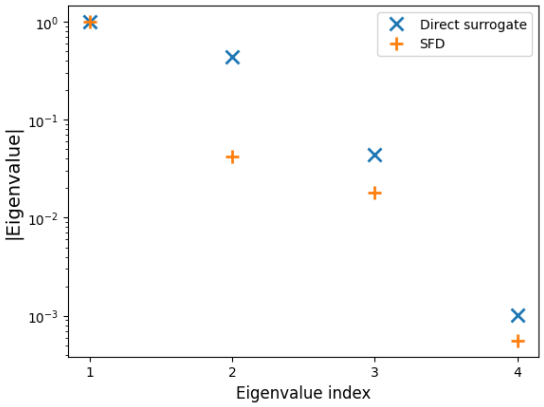
(a) $\phi(\theta, t)$



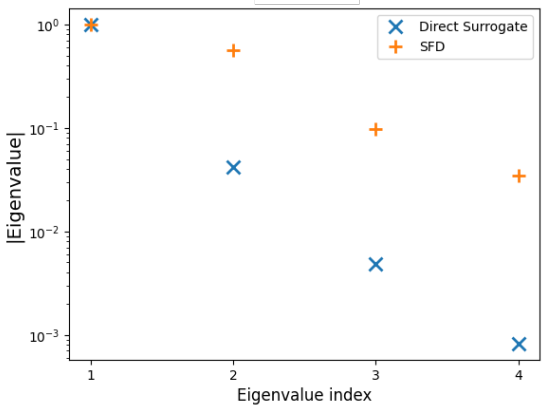
(b) HH



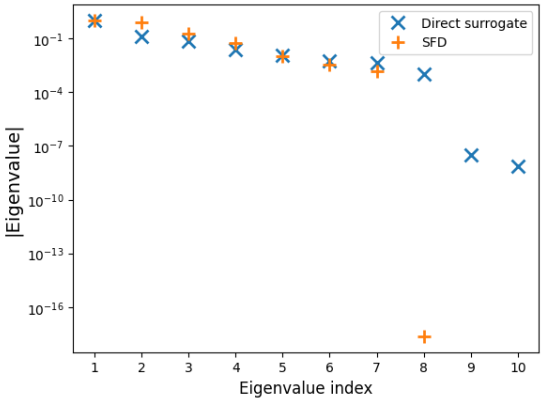
(c) Agent-based SIR



(d) BH



(e) RCK ABM



(f) Mark-0

Figure 2.22: **Eigenvalue spectrum plots for deterministic surrogates**. These spectra are each computed at a single point in parameter space, as opposed to an average over many points. We take this approach as using the averages of eigenvalues would not have a clear interpretation, because the eigenvalue ratios for a given model can differ greater across various locations in parameter space. For each model, fifteen different parameter vectors were used, each drawn from the distributions specified in Appendix C.1.

2.7.8 RMSE Values – All Models

The RMSE values that resulted from perturbing parameters in stiff/sloppy eigendirections are reported in Table 2.1. Notably, the RMSE from perturbing by the second-stiffest eigenvector (RMSE(2nd)), was not always significantly greater than the RMSE from the sloppiest perturbation, RMSE(last). This observation aligns with the results seen in the CS matrices: identification of the RCK and Mark-0 eigenvectors was more difficult, so it is unsurprising that the expected RMSE ordering does not hold.

Model	**RMSE(1st)**	**RMSE(2nd)**	**RMSE(last)**
$\phi(\theta, t)$	0.0458	0.0203	0.0078
HH	0.0417	0.0145	0.0074
Agent-based SIR	0.0354	0.0260	0.0211
BH	0.1736	0.0818	0.1442
RCK ABM	0.0255	0.0187	0.0230
Mark-0	0.0670	0.0088	0.0758

Table 2.1: **RMSE values for deterministic surrogates**, resulting from perturbations in the directions of the deterministic surrogate eigenvectors. Perturbations in the two stiffest eigendirections ("1st", "2nd") and the sloppiest eigendirection ("last") are shown. For all models except BH, we found RMSE(1st) > RMSE(2nd), as expected. For the most complex models – the RCK ABM and the Mark-0 – we found RMSE(2) < RMSE(last), which is unexpected for effective surrogates. This observation is likely a result of the difficulty deterministic surrogates have in identifying the sloppiest eigendirerctions in complex stochastic models. For each model, fifteen different parameter vectors were used, each drawn from the distributions specified in Appendix C.1. RMSE values were computed after pertrubing parameters by 8% of their initial values

2.8 Discussion

The deterministic surrogate method was shown to be effective in identifying the stiff eigendirections of some, but not all of the models considered. The effectiveness of the deterministic surrogates in replicating the sloppiness of highly nonlinear and/or nondeterministic models (such as the Agent-based SIR and the HH model) seems impressive, given that the surrogate can produce only a single estimate from any given parameter vector, while the nondeterministic models they are emulating can produce significantly different outputs for the same parameter vector. Our results show it is possible to use deterministic surrogates to identify the eigendirections of nondeterministic models, which is advantageous in cases where the model is slow to run. In particular, stiff eigendirections were most easily identified by the surrogate

networks. However, the failure of the method in the face of the most complex models we tested – the RCK ABM and Mark 0 – demonstrates the limitations of deterministic surrogates, suggesting different surrogation methods are necessary.

We will now discuss several other observations made during our research: (1) the relative time cost of the SFD method, (2) the difficulty in identifying sloppy eigendirections compared to stiff ones, and (3) the observations that surrogates may be poor emulators of a model, yet remain capable of capturing parameter sensitivity accurately. Finally, we discuss the potential need to retrain surrogate networks in new regions of parameter space, and alternative neural network architectures that could improve surrogate accuracy.

2.8.1 Time Cost of Eigenvector Computation

In the course of comparing a surrogate's eigenvectors against those of the its respective model, the speed advantage of the surrogation method was evident in the more complex models. Finding the Jacobian of a trained FNN surrogate took around 0.3 seconds in all cases, demonstrating the speed of running the surrogate forward (and running a backpropagation algorithm to determine the elements of the Jacobian matrix). For a given point in parameter space, the deterministic surrogate needed to be run only once to compute the Jacobian.[8]

	$\phi(\theta, t)$	**HH**	**SIR**	**BH**	**RCK**	**Mark-0**
Model Simulation time	0.0004	0.012	0.067	0.0002	9.8	1.0
FNN simulation time	0.031	0.03	0.03	0.03	0.03	0.029
FNN Jacobian time	0.096	0.0935	0.088	0.096	0.111	0.099
SFD samples	100	100	100	400	30	75
SFD Jacobian time	0.45	10.6	44.7	0.002	1546	1459
Speed-up factor (SFD/FNN)	4.7	11	5×10^2	2×10^{-2}	1.4×10^4	1.5×10^4

Table 2.2: **Compute time for FNN and SFD methods**. For all models except BH, FNN Jacobians could be computed significantly faster than their SFD counterparts, as measured by the "Speed-up factor", equal to SFD Jacobian time / FNN Jacobian time. SFD Jacobians were calculated using forward-differences. SFD sample sizes were reduced for the RCK and Mark-0 models, to make it computationally feasible to experiment with these models. Times shown are in seconds.

[8] A deterministic surrogate requires a single run to compute the Jacobian, while a nondeterministic generative surrogate requires multiple samples. Despite the need for multiple runs, the fast execution speed of ANNs (both in the forward direction and the backward direction) made sampling numerous times from a generative neural network very fast.

Meanwhile, computing the Jacobian of the ABMs via SFD could take tens of minutes. This sluggishness resulted from the SFD Jacobian calculations requiring numerous runs of the model. One forward-difference SFD sample required running the model at θ^* and at the p different perturbed θ values (each corresponding to perturbing a single parameter). Furthermore, multiple samples were needed to compute an accurate estimate of the average SFD Jacobian at a point, due to stochasticity. For example, computing the SFD eigenvectors at one point in four-dimensional parameter space, with 20 samples for each Jacobian, required running the model $(4 + 1) \times 20 = 100$ times. Repeating this at 15 points in parameter space required $1,500$ runs, which became expensive for models requiring one second or more to run. However, because the BH model required so little time to run (around 0.07 seconds per simulation), the FNN method did not provide a benefit over SFD. This is unsurprising, given that the aim of the surrogate method is to replace SFD derivatives when model simulation is slow and SFD calculations are time consuming.

2.8.2 Difficulty Identifying the Sloppiest Eigendirections

When using surrogates to determine eigendirections, it was often observed that the stiffest eigenvectors were most easily identified by the surrogates, with sloppy eigenvectors being more difficult to determine. Additionally, the SFD method often required far more samples for sloppy eigenvector estimates to converge, compared to the number needed to estimate the stiff directions. The reasons this added demand can arise with SFD estimation and the surrogate method are discussed below.

SFD in Presence of Gaussian Noise

Suppose we have a model f, and a function y that produces noisy realisations of f, that is, $y(\theta, t) = f(\theta, t) + \varepsilon_{\theta,t}$, where $\varepsilon_{\theta,t} \sim \mathcal{N}(0, \sigma_t^2)$ is mutually independent Gaussian noise. The one-sample SFD estimate of the derivative in direction $\delta\theta$ results from computing one realisation of $y(\theta)$ and one realisation of $y(\theta + \delta\theta)$.

$$y(\theta, t) = f(\theta, t) + \varepsilon_{\theta,t} \tag{2.15}$$

$$y(\theta + \delta\theta, t) = f(\theta + \delta\theta, t) + \varepsilon_{\theta+\delta\theta,t} \tag{2.16}$$

$$\delta y = \delta f + \underbrace{\varepsilon_{\theta+\delta\theta,t} - \varepsilon_{\theta,t}}_{=\nu} \tag{2.17}$$

$$\frac{\delta y}{\delta\theta} = \frac{\delta f}{\delta\theta} + \frac{\nu}{\delta\theta}, \tag{2.18}$$

where $\nu \sim \mathcal{N}(0, 2\sigma_t^2)$. The N-sample SFD estimate is:

$$\frac{\partial y}{\partial \theta} = \frac{\partial f}{\partial \theta} + \frac{1}{\delta\theta} \underbrace{\frac{1}{N} \sum_{i=1}^{N} \nu_i}_{=\overline{\nu}} . \tag{2.19}$$

For $\delta y/\delta\theta$ to be an accurate estimate of the derivative of f, we require that the RHS of equation 2.17 is not dominated by $\overline{\nu}/\delta\theta$. This requires sufficient SFD samples so that $\delta y/\delta\theta \gg \overline{\nu}/\delta\theta$.

The number of samples required to accurately estimate directional derivatives will depend on the size of $\partial f/\partial\theta$. If the perturbation aligns with a sloppy eigendirection, $\partial f/\partial\theta$ may be extremely small, meaning the derivative estimate will be dominated by the noise term unless a large number of samples are taken. It is possible the number of samples chosen may be sufficient for derivatives in stiff directions to be accurately estimated, but those in sloppy directions to be inaccurate.

This analysis can be extended to cases where nondeterminism is present in a non-Gaussian form. Provided that the errors are not highly positively correlated and of similar magnitude (i.e., we assume $\varepsilon_{\theta,t} \not\approx \varepsilon_{\theta+\delta\theta,t}$, even if they may be correlated), the same issue persists: accurately estimating sloppy directions requires many more samples compared to stiff ones.

Surrogates and Sloppy Eigendirections

Suppose a deterministic surrogate S has been trained on data generated from y at different values of θ. In this case,

1. Choices made regarding surrogate architecture, hyperparameters, and training strategy may lead to an imperfect surrogate, even if trained on a very large volume of data.

2. Because the training dataset is finite, the exact values of $f(\theta, t)$ will not be inferable from the set of $y(\theta, t)$ samples available. This disconnection can allow the surrogate to learn the approximate value of $f(\theta, t)$, but it will never be exact.

Therefore, we must assume a surrogate S has a training error, η:

$$S(\theta, t) = f(\theta, t) + \eta_{\theta,t}. \tag{2.20}$$

Even if the data generation process has zero noise ($\varepsilon_{\theta,t} = 0, \forall\theta, t$, so $y(\theta, t) = f(\theta, t)$), training error will persist to some degree.[9] Consequently, differentiation of the surrogate in the direction $\delta\theta$ will be imperfect:

$$\frac{\delta S}{\delta\theta} = \frac{\delta f}{\delta\theta} + \frac{\eta_{\theta+\delta\theta,t} - \eta_{\theta,t}}{\delta\theta}. \tag{2.21}$$

2.8.3 Consistent Sensitivities, Inconsistent Simulations

There were instances where the trained surrogate could predict stiff/sloppy eigendirections, but not accurately simulate the model of interest. We sometimes observed this phenomenon with the BH model, as illustrated in Figure 2.16. The time series generated from the original model and the surrogate were very different from one another, but their stiff eigenvectors were relatively closely aligned. The reason for the discrepancy between the outputs of a model and its surrogate may be the relationship between the training error at different points in parameter space. Examining equation 2.21, one can see that if there is a high correlation in direction and magnitude of training errors (i.e., cases where $\eta_{\theta+\delta\theta,t} \approx \eta_{\theta,t}$), the derivatives of the surrogate and the underlying model may be relatively close, even though the surrogate is imperfect.

Alternatively, the main effect of stiff and sloppy movements in parameter space may correspond to changes which primarily affect the average of the model's output, rather than the overall shape of the output, in which case an inability to accurately replicate oscillatory behaviour may not inhibit accurate identification of stiff and sloppy eigendirections (as was the case with the BH model). However, for some models, the parameters may influence behaviour in the opposite way to this, in the sense that the average has little dependence on the parameters, and movements in parameter space only affect the frequency, amplitude, and/or phase of oscillations about this average. In the next chapter, we start by examining such a model (which we name $H(\theta, t)$), and demonstrate that a deterministic surrogate cannot identify its stiff and sloppy eigenvectors, even though the surrogate can learn the mean behaviour without difficulty.

The technique developed in this chapter did not fail in the face of this phenomenon when applied to the BH model, as the stiffest eigendirection could still be identified. However, it is possible that it will fail for models with greater variation between

[9] An exception would be the very rare event that that some point during training, the network weights happen to take the exact values – to the limits of machine precision – such that the surrogate network becomes a perfect emulator of the data generating function. However, for non-trivial data generating processes, this is incredibly unlikely.

runs. For this reason, it is desirable to consider nondeterministic surrogate structures that are naturally more suitable for nondeterministic models with a high degree of variation.

2.8.4 Exploring Large Parameter Spaces Using Surrogates

Suppose parameters are sampled from a large region of parameter space, and these samples have good coverage of that space. Exploring parameter space by following stiff eigendirections may lead us to region(s) of parameter space outside the training set, and thus we may find ourselves in a region of parameter space where the surrogate may be incapable of emulating the target model accurately. In this case, it may be important to re-train the surrogate in the new region of parameter space that has been reached, or SFD could be used from there for further exploration. While training a new surrogate could require significant time, the discovery of a new region of parameter space may be highly useful in itself, potentially aiding the construction of a "map" of phases and stiff/sloppy directions throughout a large section of the parameter space of the model.

2.8.5 Alternative Architectures: Recurrent Neural Networks, LSTMs, and GRUs

Given the models under consideration produce time series outputs, it seems natural to use a recurrent neural network for the task of building surrogates. A simple long short term memory (LSTM) surrogate was tested on the models above, but no significant performance improvements were found. It seems possible that networks with a more sophisticated structure – such as gated recurrent unit (GRU) networks – could be superior, but for the problems we have considered, these are unnecessary.

It is possible models with more complex nonlinear behaviours (e.g., complex, large scale macroeconomic ABM, such as that of Poledna et al., 2023 [118]) will require more complex surrogates, implying that other deterministic surrogate architectures may be worth contemplating. However, the development of nondeterministic surrogates seems more worthwhile, as these are likely to better handle large "nondeterministic nonlinearities" present in some ABMs, which can inhibit the effectiveness of deterministic surrogates.

2.9 Chapter 2 Conclusion

This chapter has demonstrated deterministic neural network surrogates attempting to directly mimic a model can successfully identify that model's stiff and sloppy directions, across a range of models. This technique is applicable more generally than the IRF technique of Chapter 1, since it does not rely on the existence of the concept of an "impulse" within a function (e.g., the BH ABM, which does not feature exogenous shocks). While deterministic neural network surrogates generally are successfully deployed to identify the stiff eigendirections across various models, we must note that the identification of eigenvectors – especially sloppy ones – was not always accurate, especially in the case of the BH model. Simple FNN architectures were found to perform sufficiently in many cases, despite the expectation that recurrent networks would more effectively emulate the time series models.

In Chapter 3, we investigate the use of nondeterministic surrogates to compute model sloppiness. These surrogates can outperform deterministic surrogates when the parameters only determine the model oscillates around a constant mean path, or when a very high degree of variability is present in a model's output when it is repeatedly run with the same input parameters.

Chapter 3

Generative Neural Networks as Surrogates

The previous chapter applied neural networks to create deterministic surrogates. This method is able to predict parameter sloppiness across a variety of models, and demonstrates particular strength in identifying stiff directions. However, an intrinsic problem with these surrogates is their inability to emulate nondeterminism in models. While this was not a serious problem for simple models with relatively little stochasticity around a mean path (e.g., the agent-based SIR model), this becomes problematic for some models with greater stochasticity – especially when a model, run repeatedly with the same parameter vector (θ), can generate very different outputs, including oscillations with different amplitudes and starting phases.

This chapter addresses the limitations of deterministic surrogates by investigating the use of nondeterministic "generative" surrogates. To build generative surrogates, we use autoencoder networks – specifically, conditional variational autoencoders (CVAEs). This is due to their ability to produce outputs that are quantitatively comparable to those of nondeterministic models, and their ability to produce outputs conditioned on a parameter vector (rather than outputs that resemble those produced from the model of interest, but without any dependence on the parameters used to produce those outputs).

An Example

To highlight the need for nondeterministic surrogates, consider the following sinusoidal model, which takes a two-element parameter vector ($\theta = [\theta_0 \ \theta_1]^T$) as an argument.

$$H(\theta, t) = \theta_0 \varepsilon_0 \sin(\theta_1 t + \varepsilon_1) \tag{3.1}$$

$$\varepsilon_0, \varepsilon_1 \sim U(0, 2\pi). \tag{3.2}$$

This function H varies in amplitude (controlled by θ_0) and phase (θ_1). For each realisation of the function, two random variables are drawn, ε_0 and ε_1, which influence the amplitude and phase, respectively. This function can be considered stochastic, as each time it is run, the output will be different, even if θ_0 and θ_1 are unchanged.

The derivatives of H are given by:

$$\frac{\partial H}{\partial \theta_0} = \varepsilon_0 \sin(\theta_1 t + \varepsilon_1) \tag{3.3}$$

and

$$\frac{\partial H}{\partial \theta_1} = \theta_0 \varepsilon_0 t \cos(\theta_1 t + \varepsilon_1) \tag{3.4}$$

The mean of each derivative over the distributions of ε_0 and ε_1 is zero:

$$\left\langle \frac{\partial H}{\partial \theta_0} \right\rangle_{\varepsilon_0, \varepsilon_1} = 0 \tag{3.5}$$

$$\left\langle \frac{\partial H}{\partial \theta_1} \right\rangle_{\varepsilon_0, \varepsilon_1} = 0. \tag{3.6}$$

When averaged over many runs (each run having different values drawn for ε_0 and ε_1) the mean is zero (as shown in Figure 3.2). Because of the zero mean at each point in time, a deterministic surrogate trained on data from H predicts[1] $H(\theta, t) = 0 \quad \forall \theta, t$ (illustrated in Figure 3.3). The sensitivity of the function w.r.t. θ is lost, meaning the deterministic surrogate will be unable to identify stiff and sloppy eigendirections.

[1]This assumes the training loss function used is symmetric, i.e., overestimates and underestimates are penalised equally.

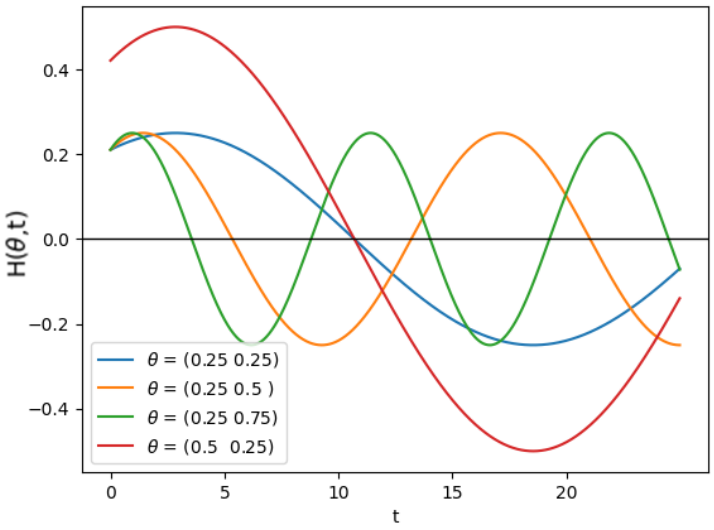


Figure 3.1: **Runs of $H(\theta, t)$, with $\varepsilon_0, \varepsilon_1$ held constant**. Different parameter combinations used, with the random variables held constant ($\varepsilon_0 = 1$, $\varepsilon_1 = 1$). The shape (amplitude and phase) of the function varies significantly with θ.

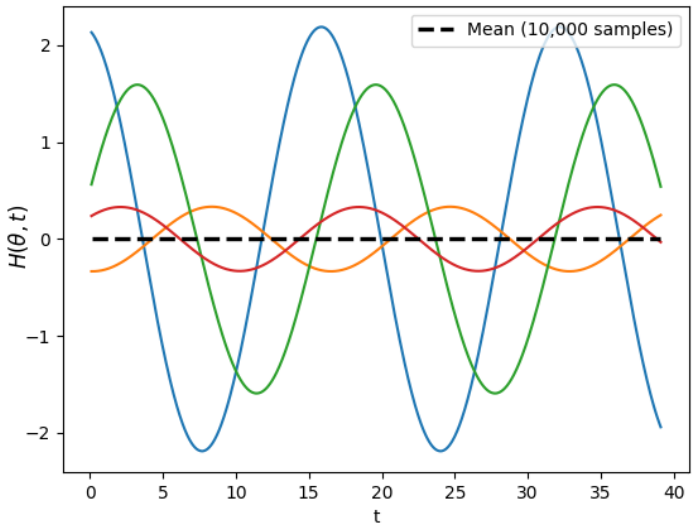


Figure 3.2: **Runs of $H(\theta, t)$, showing zero mean**. Plot shows four realisations of the nondeterministic, oscillatory function H, with fixed parameters ($\theta_0 = 0.5$, $\theta_1 = 0.75$). The mean of 10,000 realisations (dashed black line) is also shown, generated using the same parameters. The function's mean is independent of θ, making it impossible for a deterministic surrogate to identify parameter sensitivity, even though the shape (amplitude and phase) of the function varies greatly with θ.

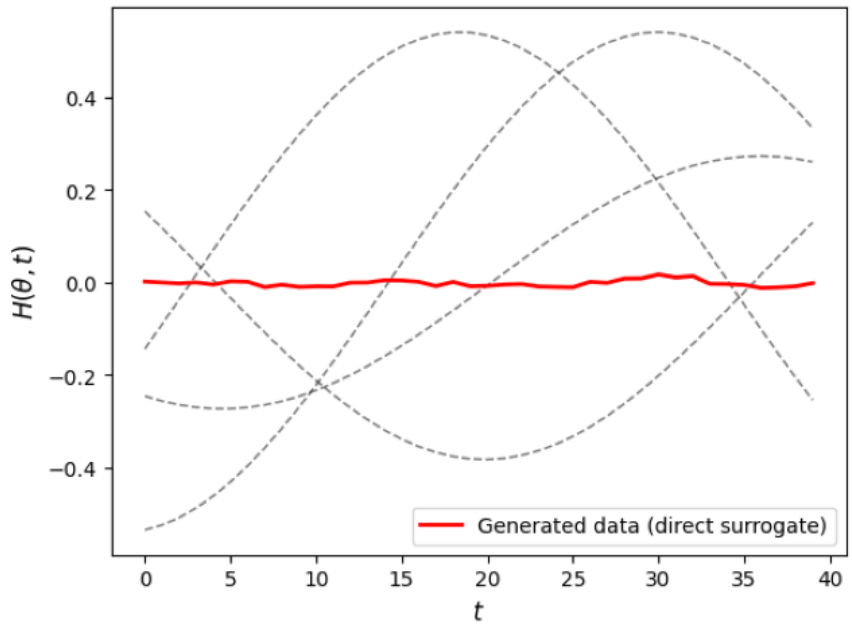


Figure 3.3: **Deterministic surrogate's output for** $H(\theta, t)$, along with four realisations of $H(\theta, t)$ with θ held constant ($\theta_0 = 0.3$, $\theta_1 = 1$). The deterministic nature of the FNN surrogate leads to its output being approximately equal to the mean value of the function (i.e., $\langle H(\theta, t)\rangle_{\varepsilon_0,\varepsilon_1}$, which is zero for all t).

To construct surrogates that can replicate nondeterminism in models, we create generative neural networks. Below, we define and describe several classes of generative models, and their applications. We then go on to describe our specific use of generative surrogates.

3.1 Generative Models

For our purposes, we define a generative model as a statistical model capable of producing outputs with a probability distribution similar to that of the outputs of the nondeterministic target function (f). A *conditional* generative model has a corresponding function f_G, which allows us to generate outputs ($f_G(\theta)$) whose distribution is statistically comparable to the distribution generated by f when applied to the same parameter vector, θ.[2]

[2]Unconditional generative models exist, such as simple Variational Autoencoders and Generative Adversarial Networks. These are also capable of generating "realistic" outputs, but cannot be conditioned on parameters (labels) used to generate the training data. i.e., an unconditional model f_U does not take an input parameter vector when generating outputs. Such models cannot easily be used as surrogates for parameterised functions.

Once trained, most generative surrogates, including the CVAEs used in this chapter, directly produce output data, rather than parameters of a probability distribution function of the possible outputs. For example, our CVAE surrogate for the SIR model generates an infection rate time series, as does the SIR model itself. However, these outputs are effectively samples from the probability distribution function ($P(X|\theta)$) learned from the training data, meaning the trained surrogate has an implicit probability density function which could be inferred from the data it generates.

Generative models can produce new data that follows the distribution of the training data, without simply duplicating the training examples. This is a useful property allowing the generation of synthetic data for varied applications, including creating surrogates for computational fluid dynamics [67], generating artificial medical data [146, 46] (which can remove the need to use private patient data when training separate diagnostic models), and augmenting "real-world" datasets with synthetic data, to provide more samples for training other AI models [61]. Generative models can also be used to create new data sets for human learning – for example, creating new X-ray images of bone fractures, for training radiologists.

To create generative surrogates, we use autoencoders – specifically, CVAEs. Our surrogates are designed to emulate the full time series outputs of the target models, rather than summary statistics computed from those models' outputs. In the sections below, we will discuss the history of autoencoders, their different structures, and existing applications. We will then describe our novel method in detail, and explain how it extends the existing applications of CVAEs as surrogates.

3.2 Literature Review: Autoencoders

An autoencoder is a type of ANN, capable of learning efficient low-dimensional representations of high dimensional data [108, 15]. The aim of a simple autoencoder is to accept high-dimensional data y, convert that data into a low-dimensional ("encoded") form, then accurately reconstruct the original data (y) from that encoded representation.

3.2.1 Basic Structure

A simple autoencoder can take an M-dimensional vector as an input – say, a length M time series – and is intended to produce an M-dimensional output approximately equal to this input. A basic autoencoder (as shown in Figure 3.4) has simple fully connected feedforward layers, and the dimension of the input layer (i.e., how many

nodes are present) is the same as that of the output layer. Importantly, one intermediate hidden layer has a relatively small number (n_L) of nodes, called the *latent* layer, or the *bottleneck*. The values of these n_L nodes are sufficient to represent the input in such a way that the rest of the network (the decoder) can approximately reconstruct the full M-dimensional data. The network before the bottleneck is known as the *encoder*, and the network after the bottleneck is the *decoder*. The possible values of the latent layer's nodes may take the form what is known as the *latent space*, typically with a far smaller dimension than that of the input.

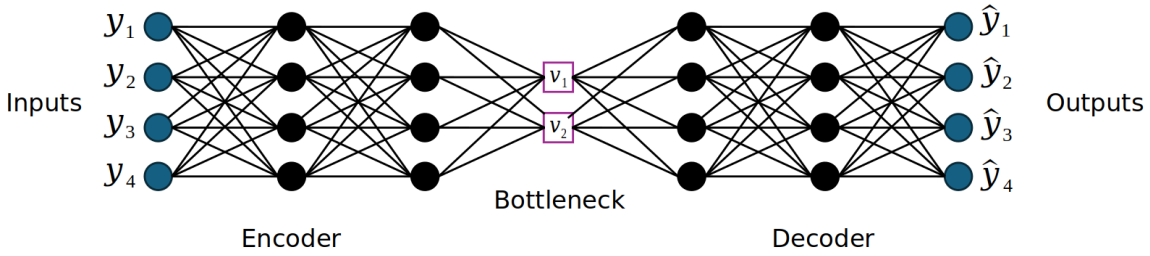


Figure 3.4: **Simple autoencoder network**, consisting of two bottleneck nodes (ν_1, ν_2), and encoder/decoder networks each with two feed-forward hidden layers each. We expect the inputs and outputs to be similar ($\hat{y}_i \approx y_i$) if the network is successfully trained.

Simple autoencoders are trained using unsupervised learning. The pairings of inputs y and their latent representations ν are not specified in advance (i.e., the training data does not consist of raw data values and corresponding latent node values) and are instead learned by the network.

3.2.2 History of Autoencoders

Autoencoders were introduced as a generalised form of principal component analysis, making use of the ability of ANNs to learn nonlinear relationships in data [90]. Initially, their primary use was to reduce the dimensionality of data, by associating each high-dimensional input with the values of the latent layer's nodes. The encoder network could also be used for feature engineering: the latent representation contains the most important information about a raw input, and this representation can be used when training other machine learning models (where a smaller-dimensional input may be desirable, to reduce the number of nodes and trainable parameters of the model).

Variational autoencoders (VAEs) were introduced in 2013 [87]. These networks incorporate stochasticity into the autoencoder architecture. Unlike with simple autoencoders, the nodes of the VAE bottleneck are treated as random variables. The encoder network maps each data input to a multivariate probability distribution (usually Gaussian) from which values for the bottleneck can be drawn. Each random draw produces a different output when fed through the decoder network, allowing a VAE to be used as a generative model.

CVAEs are a further evolution of the autoencoder architecture, introduced in 2015 [138]. Unlike standard VAE, a CVAE can predict a set of possible values of a nondeterministic function g, conditioned on a parameter vector θ (also known as a "label"). Once a CVAE is trained, a new parameter vector θ' can be used to generate a new outputs y', which have an approximately similar distribution to the outputs of $g(\theta')$. In other words, a CVAE output y' is a plausible realisation of $g(\theta')$, meaning the CVAE can act as an emulator for g. Unlike VAEs, supervised learning is used to train CVAEs: training pairs (θ, y) are provided, rather than y alone.

3.2.3 Autoencoder Architectures

Below, we describe the network architectures of simple autoencoders, VAEs and CVAEs. The architectures described are illustrated in Figure 3.5.

Simple Autoencoder Structure

A simple autoencoder can consist of several hidden densely-connected feedforward layers, a bottleneck hidden layer, then more hidden feedforward layers before the output. Feedforward layers can be replaced with convolutional or recurrent layers.

Training is carried out using a simple objective function measuring the "reconstruction loss", usually using the L_2 norm:

$$\text{Reconstruction Loss} = \|y - \hat{y}\|_2. \tag{3.7}$$

Note that if the data is categorical in nature (e.g., identifying hand-drawn digits), loss functions are commonly used, such as binary cross-entropy.

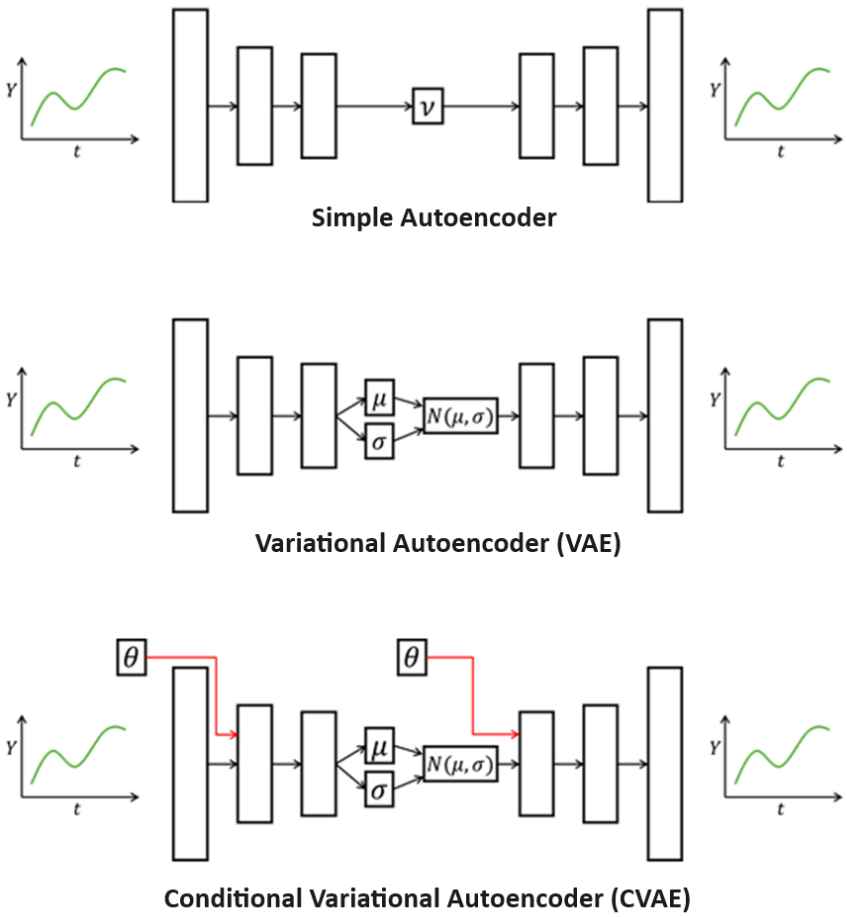


Figure 3.5: **Types of autoencoder**. In all cases, the networks are trained in such a way that the input time series (Y) is approximately equal to the output. Vertical rectangles represent fully connected hidden feedforward layers of the neural networks. (Adapted from Figure 1 of Yonekura & Suzuki [160].)

VAE Structure

In a VAE, the latent space is a multivariate distribution, which we assume to be Gaussian. The training process seeks to achieve two goals: minimise the reconstruction loss, and minimise the KL divergence between the latent distribution and a chosen prior distribution (usually the multivariate standard normal, $\mathcal{N}(0, I)$):

$$L_V = \text{Reconstruction Loss} + \text{KL Divergence}, \tag{3.8}$$

where $q(\nu|y)$ is the distribution of the latent variables given input data y, and $p(\nu)$ is the latent variables' prior distribution.

If the loss function consisted solely of the reconstruction error, training could lead to the elimination of all stochasticity: the standard deviations (σ) could collapse to

zero, and the means (μ) would take the (possibly unique) values best associated with optimal reconstruction. Effectively, the VAE would become a simple (non-stochastic) autoencoder. The inclusion of KL divergence also has a regularising effect.

CVAE Structure

A CVAE uses training pairs, (θ, Y). In our context, θ is a parameter vector, and Y is a time series realisation resulting from feeding θ into the model of interest. However, in other applications, θ could take values from a discrete set – such as the set of digits $0-9$, each associated with handwritten images of the corresponding digits[3].

To create a CVAE, the parameter vector θ is concatenated with the corresponding data vector Y. This concatenated vector is used as the input of the CVAE. The parameter vector θ is also concatenated with the latent variables (ν) to create a vector injected into the network at the bottleneck stage. The output of the CVAE is the data (Y). The loss function used during training is similar to that used for a VAE (equation 3.8), but contains additional conditioning, based on the parameters θ:

$$\begin{aligned} L_C &= \text{Reconstruction Loss} + \text{KL Divergence} \\ &= \mathbb{E}_{q(\nu|y,\theta)}[\log p(y|\nu,\theta)] - D_{KL}(q(\nu|y,\theta) \parallel p(\nu|\theta)). \end{aligned} \tag{3.9}$$

3.2.4 Existing Applications of CVAE Networks

In this section, we describe several ways CVAE networks have been used in multiple fields to solve a variety of complex tasks. The flexibility of CVAEs has made them popular in a number of different areas, including scientific research, time series forecasting, anomaly detection, and creative design processes.

CVAEs as Surrogates

While deterministic surrogates can be used for some applications, it is often desirable to use generative models. For example, in weather modelling, small uncertainties in initial conditions can lead to drastically different future states [101], meaning the are many plausible future paths which must be forecast.

CVAEs have seen significant use as generative surrogates across a range of fields. They have been used widely in fluid dynamics, where the computational expense of solving the underlying physical equations (as discussed in Section 2.1.1) makes

[3]A seminal example of the data behind this problem is the MNIST database of handwritten digits.

it desirable to build efficient surrogates. Examples of this include Gundersen et al. [67], who create a CVAE variant (a "semi-conditional variational autoencoder") to simulate fluid flows based on limited data, and estimate uncertainty in the surrogate's output. Lukošiute et al. [103] use CVAEs to model collisions between neutron stars and black holes, which involves modelling the flow of plasmas at extremely high temperatures. Other authors have used CVAE surrogates for design problems in engineering. For example, Yonekura and Suzuki [160] generate designs for aerofoils and turbine blades. Additionally, Balmer et al. [13] use CVAEs to find pedestrian bridge designs that fit performance criteria, as an alternative to manual iterative techniques which are often highly time-consuming. In these contexts, surrogates are used to solve inverse problems: beginning with a design goal (such as reducing a bridge's strength-to-weight ratio), a geometric design must be found to fulfil that goal, as opposed to the problem of computing the structural properties of a specified geometric design using known laws of physics. CVAEs are also capable of forecasting the future values of time series – such as the price of a firm's shares over time – given historial data up until that point in time. In such an application, the conditioning vector θ can consist of the historical values of that time series (and any relevant external variables, such as industry classifications of the share's respective firm). One such example is trajectory prediction: Gomez-Gonzalez et al. [59] use a modified CVAE to generate real-time predictions of the trajectories of physical objects, which they use to enable a robot to play simple ball games.

3.2.5 Other Generative Methods

A range of generative neural network architectures have been applied to problems in a variety of fields [32]. For many applications – including text generation, image generation, audio synthesis and time series modelling – models other than autoencoders have become prominent. These include generative adversarial networks, LSTMs, transformers, and diffusion models [70]. For our purposes, we could use LSTMs, which are often capable of capturing long-range dependencies. However, these networks generally produce unique outputs, and are not as well-suited to our problem of stochastically generating many possible realisations.

Generative techniques that do not rely on neural network structures include hidden Markov models, Gaussian mixture models, variational Bayesian methods, Dirichlet process models, and stochastic autoregressive models [70].

3.3 Our Generative Surrogate Method

In this section, we outline how our CVAE networks are constructed and trained. The CVAE networks were implemented using the Keras software library [35].

Constructing the CVAE and Training Data

Below is a description of how the CVAE surrogates were constructed, including the structure of the networks' components. The precise structure of the networks used for each model can be found in Appendix D.1.

1. **Input Layer**: a node for each time series output point (i.e., a time series of length T requires T input layer nodes).

2. **Encoder sub-network**: several densely-connected feedforward layer were used. Typically, this sub-network consisted of five to eight layers, with each layer containing between 32 and 128 nodes, with the smallest layer being immediately before the latent layer.

3. **Latent Space Design**: a small number (typically two or three) nodes were used for the latent layer. When generating new time series outputs, these nodes took values drawn from a multivariate distribution with a diagonal covariance matrix. The mean and variance of this distribution were learned during the training stage.

4. **Decoder sub-network**: this consisted of feedforward layers. In most cases, we used the same layer sizes and activations as those used in the encoder sub-network, but in the reverse order.

5. **Loss function**: as described in section 3.2.3, the loss function is a linear combination of the reconstruction loss (L_2 norm) and a KL divergence term, with a standard normal distribution as a prior.

6. **Training set**: Around 1,000 - 2,000 runs of the target model were generated to serve as training and validation sets (80% for training, 20% for validation). Parameters vectors (in their normalised form) were drawn from the multivariate uniform distribution $[0, 1]^p$.

A diagram illustrating the full structure of the network used for the SIR model can be found in Figure D.1 in the appendix.

Training the Network

This step is the same as the one used in the creation of deterministic surrogates (Chapter 2). Specifically, the steps involves are:

1. Sample from the parameter space N times, producing parameter vectors $\Theta^1, \Theta^2, \ldots, \Theta^N$.
2. Run the model on those N parameter vectors to produce time series $G^1, G^2, \ldots, G^N$.
3. Train the CVAE, using pairs (G^k, Θ^k) as training examples.

Computing the FIM

As described above, a trained CVAE will be capable of taking in a parameter vector θ and generating many new output runs, y_i. Unlike FNNs – which require just one backpropagation run to compute the Jacobian – CVAE surrogates require a multiple samples to be taken from the latent space, with backpropagation applied to each of these samples. If a appropriate network architecture was chosen (and sufficient training data was provided), the following process can be used to compute the Jacobian of a trained network.

1. A new parameter vector θ^*, is chosen, which represents the point in parameter space where we wish to find the stiff/sloppy eigendirections.
2. Two vectors are provided to the decoder section of the CVAE: the new parameters θ^*, and a sample z_s from the standard normal distribution. This provision allows the decoder to produce output data (i.e., a single new time series). Through backpropagation, the derivative of each output w.r.t. the elements of θ^* is computed, giving us a Jacobian matrix J_s.
3. The process is repeated many times: a new sample is chosen ($z_{s'}$), and this is run through the decoder with the same parameter set θ^*. The decoder then generates more outputs, and more $J_{s'}$ values.
4. By averaging the J_k values, a new Jacobian matrix is produced (J_{mean}). This matrix can be used to compute the approximate FIM ($FIM_{CVAE} \approx J_{mean}^T J_{mean}$), from which stiff and sloppy eigenvectors can be computed.

If enough samples are taken from the latent space, and an architecture was chosen such that output generation is fast,[4] the eigenvectors of the model can be computed quickly. This was found to be the case with the CVAEs tested, and the process was far quicker than using SFD for models of significant complexity.

[4]All CVAEs tested could produce generative outputs within a few seconds.

3.4 Results

To evaluate the performance of our generative surrogation method against the performance of deterministic surrogates, we compared the CS matrices and RMSE value results. In general, we found that generative surrogates for our most complex models – namely, the RCK ABM and Mark-0.

The results for each model are presented below. Our primary focus is the comparison of the cosine similarity matrices derived from deterministic surrogates against those generated from the CVAE surrogates. Notably, even when the runs generated by our CVAEs did not look particularly similar to the target model's output (which happens reasonably often), the stiff or sloppy directions, or both, are often well-aligned with the SFD-derived directions. The structure and hyperparameters used to construct and train the generative surrogates is listed in Appendix D.1.

3.4.1 HH Results

In the case of the HH model, our CVAE surrogate method accurately recovered the first three eigenvectors of the model, as shown in Figure 3.1. However, the CVAE surrogate did not lead to a significant improvement over deterministic surrogates, likely because of the relatively small amount of (Gaussian) stochasticity in the model.

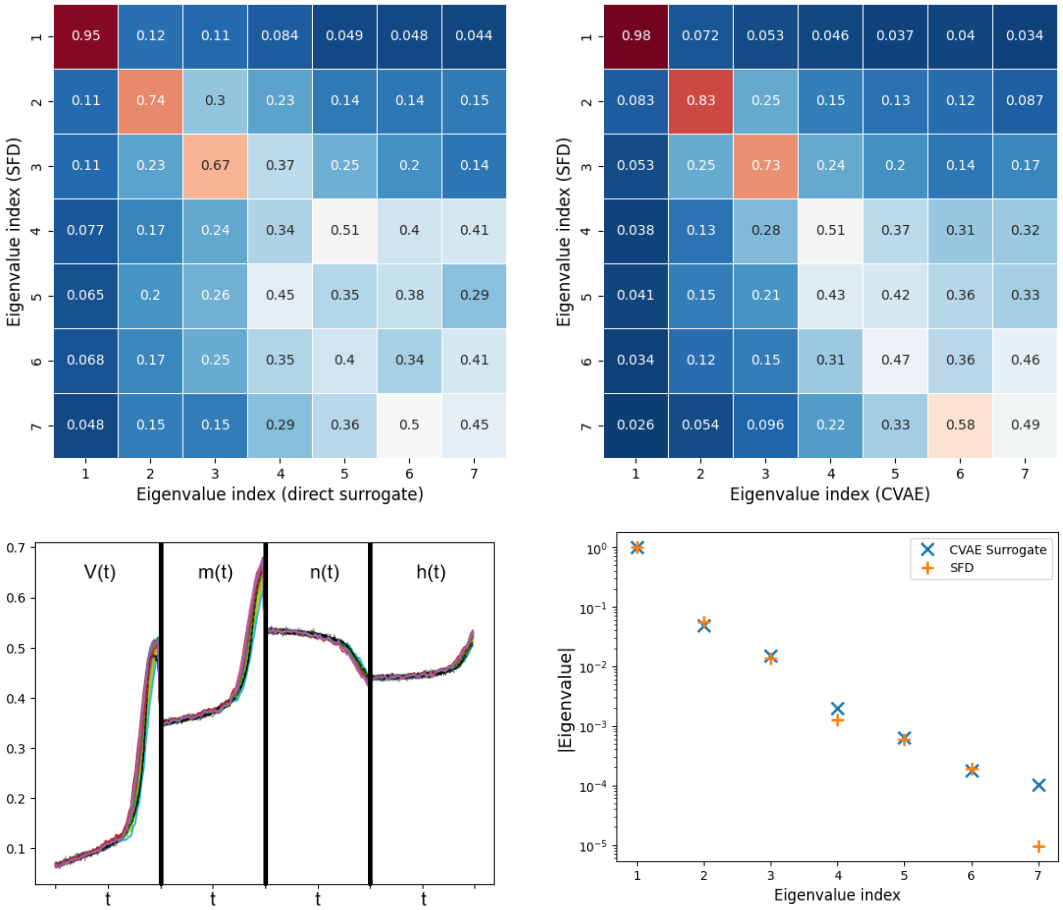


Table 3.1: **HH CVAE surrogate performance**. Clockwise from top-left: (1) deterministic surrogate CS, (2) generative surrogate CS, (3) eigenvalue plot (one run), and (4) example outputs from the generative surrogate (all four HH output channels). The generated outputs (coloured lines) are virtually indistinguishable from the HH output (black). A small improvement in alignment was observed in the CS matrix. Fifteen different parameter vectors were used for each experiment, each drawn from the distribution specified in Appendix C.1.2.

3.4.2 Agent-based SIR Results

Our deterministic surrogate method was successful in accurately identifying all four eigenvectors of the SIR model, leaving little room for improvement. The CVAE surrogate performance is similar. The runs generated by the CVAE were visibly close to those generated by the SIR model.

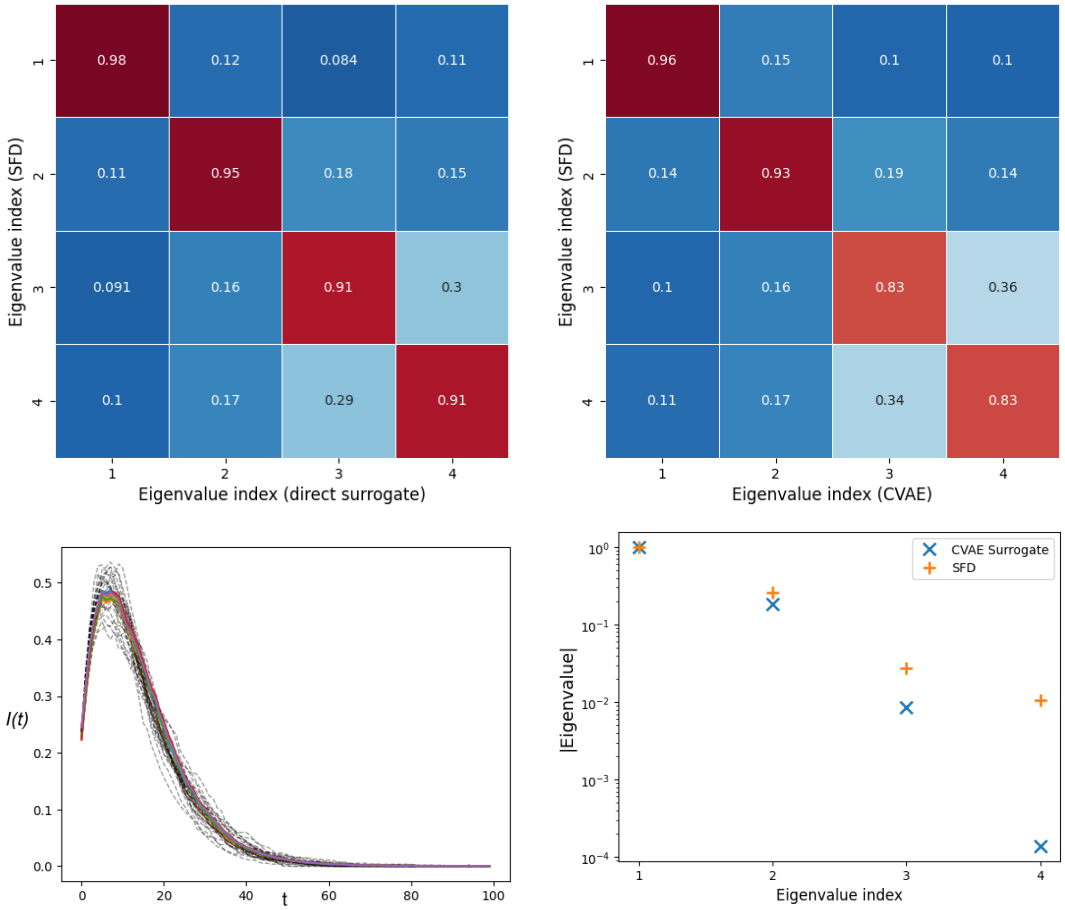


Table 3.2: **Agent-based SIR CVAE surrogate performance**. Clockwise from top-left: (1) deterministic surrogate CS, (2) generative surrogate CS, (3) eigenvalue plot (one run), (4) example outputs from generative surrogate (coloured) and outputs from the model itself (black), with parameters unchanged. The deterministic surrogate and CVAE cosine similarity matrices are similar, suggesting the CVAE method has not yielded an improvement over deterministic surrogation. Fifteen different parameter vectors were used for each experiment, each drawn from the distribution specified in Appendix C.1.3.

3.4.3 BH ABM Results

Using deterministic surrogates, accurate identification of the stiff/sloppy directions was challenging – especially for those other than the stiffest eigenvector. Our CVAE method performed substantially better: the stiffest eigenvector was identified more accurately, and the other eigenvectors could be reasonably well identified (cosine similarities between 0.7 and 0.87). This result usefully demonstrates the ability of generative surrogates to outperform deterministic ones (although the performance gain in this case is small).

Although the CVAE eigenvectors aligned well with those computed by the conventional SFD method, the time series produced by the CVAE showed noticeably smaller amplitudes of oscillation compared to those from the BH model, indicating the CVAE surrogate was not acting as a reliable emulator. However, the CVAE was still able to capture the model's sensitivity to parameter changes (a phenomenon seen in Chapter 2, and discussed in section 2.8.3), unlike the deterministic surrogate.

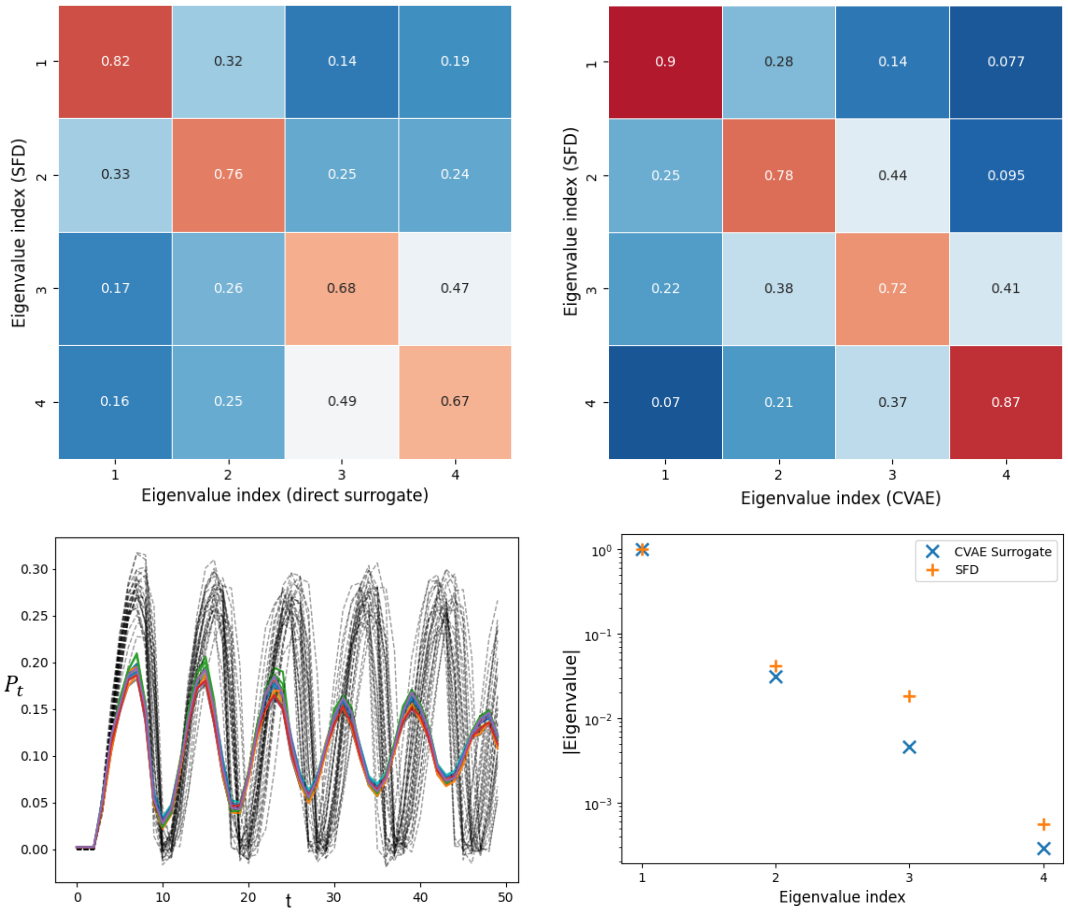


Table 3.3: **BH CVAE surrogate performance**. Clockwise from top-left: (1) deterministic surrogate CS, (2) generative surrogate CS, (3) eigenvalue plot (one run), and (4) example outputs from generative surrogate (coloured) and outputs from the model itself (black), with parameters unchanged. The CVAE performance was slightly better than the deterministic surrogate, as measured by the magnitude of the diagonal elements of the CS matrices. Fifteen different parameter vectors were used for each experiment, each drawn from the distribution specified in Appendix C.1.1.

3.4.4 Agent-based RCK Results

In the case of the RCK model, the CS matrix indicated more alignment occurred between CVAE and SFD eigenvectors (shown in Figure 3.4). The stiffest CVAE and SFD eigenvectors were better aligned, and the remaining eigenvectors displayed a significant degree of alignment (although swapping did often occur between the second and third stiffest eigenvectors). As was the case for the BH model, the outputs generated by the CVAE surrogate did not show oscillations as large as those generated by the RCK model itself.

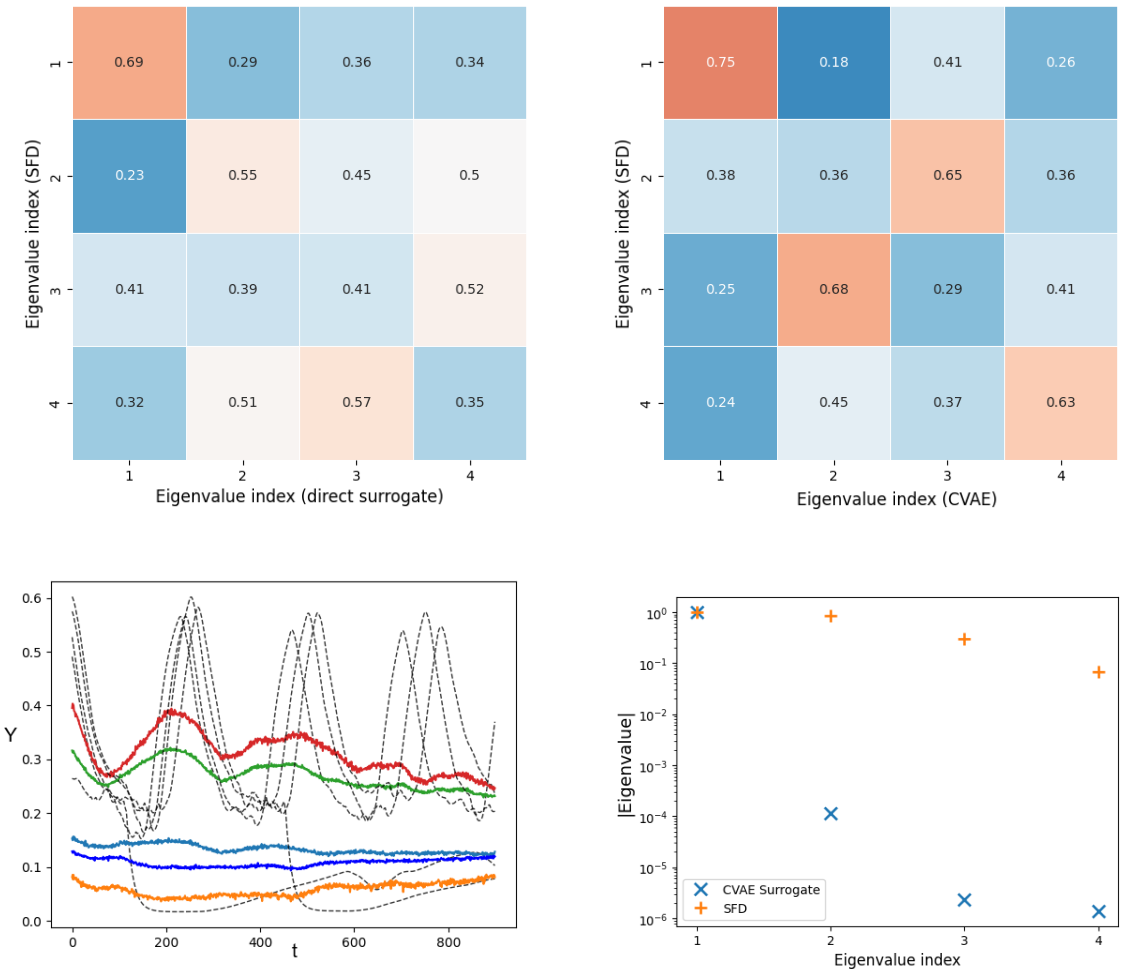


Table 3.4: **RCK ABM CVAE surrogate performance**. Clockwise from top-left: (1) deterministic surrogate CS, (2) generative surrogate CS, (3) eigenvalue plot (one run), and (4) five example outputs from generative surrogate (coloured) and outputs from the model itself (black), with parameters unchanged. As indicated by the CS matrices, the CVAE has performed better at identifying the model's eigendirections. Fifteen different parameter vectors were used for each experiment, each drawn from the distribution specified in Appendix C.1.5.

3.4.5 Mark-0 Results

In Chapter 2, we observed that the deterministic surrogate method completely failed to identify any of the stiff/sloppy directions of the Mark-0. Our CVAE surrogate performed significantly better than our deterministic surrogate, with the stiffest eigenvector being recovered reliably, and the next two eigenvectors being somewhat identifiable. As was the case with the BH model, the time series outputs of the CVAE were noticeably different from the Mark-0's outputs, with far smaller oscillations observed.

While it was difficult for the CVAE to identify the eigendirections accurately (beyond the first two), its performance is notably superior to that of the deterministic surrogate, which was unable to identify any eigendirections of the Mark-0.

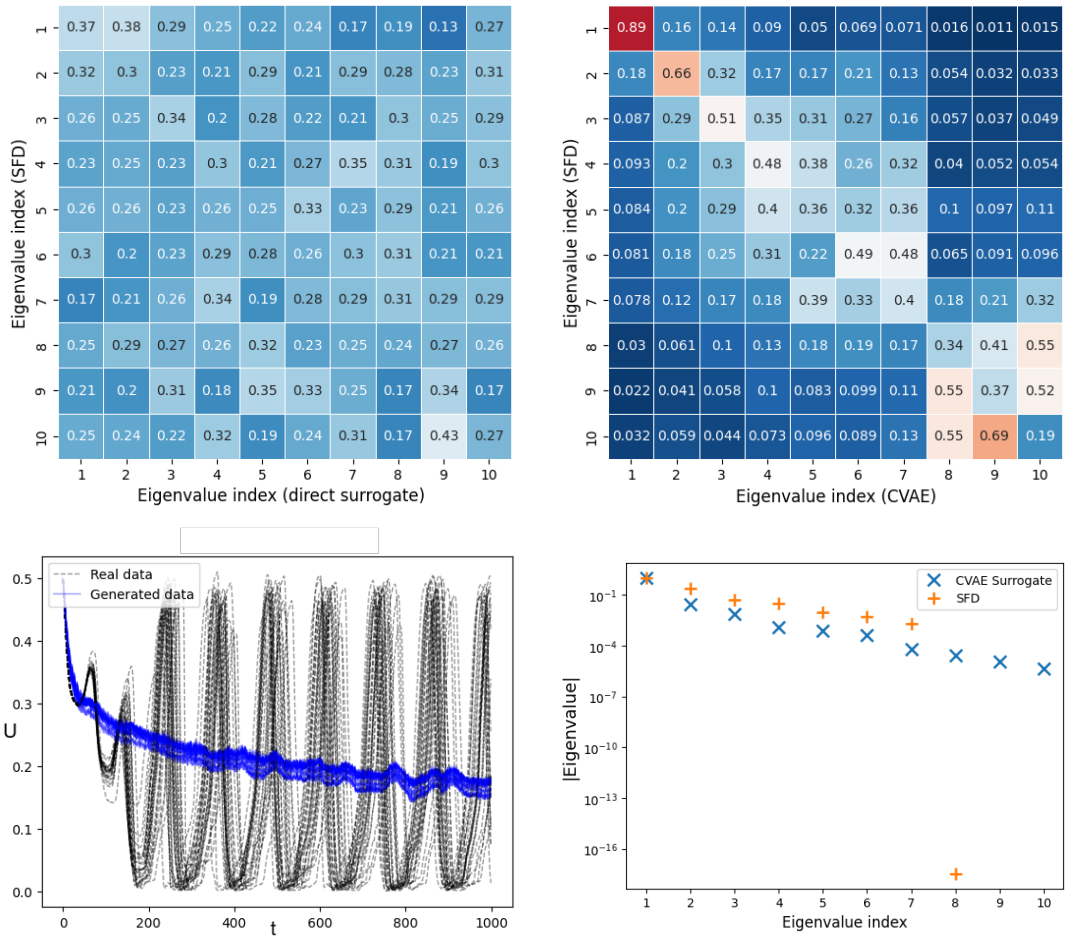


Table 3.5: **Mark-0 CVAE surrogate performance**. Clockwise from top-left: (1) deterministic surrogate CS, (2) generative surrogate CS, (3) eigenvalue plot (one run), and (4) example outputs from generative surrogate (coloured) and outputs from the model itself (black), with parameters unchanged. Although the CVAE was unable to accurately identify many of the eigenvectors, its performance was substantially better than that of the deterministic surrogate (as measured by CS matrices). Fifteen different parameter vectors were used for each experiment, each drawn from the distribution specified in Appendix C.1.6.

3.4.6 RMSE Values – All Models

The RMSE values resulting from perturbing parameters in stiff/sloppy eigendirections are reported in Table 3.6. In most cases, the expected ordering (RMSE(1st) > RMSE(2nd) > RMSE(last)) holds, with the exception of the Mark-0. Each RMSE value is the average of 15 runs.

Model	RMSE(1st)	RMSE(2nd)	RMSE(last)
HH	0.1251	0.079	0.0528
Agent-based SIR	0.0235	0.0163	0.0118
BH	0.1702	0.0569	0.1295
RCK ABM	0.0244	0.0221	0.0184
Mark-0	0.2054	0.2275	0.2029

Table 3.6: **RMSE values for CVAE surrogates**, resulting from perturbations in the direction of the two stiffest eigendirections ("1st", "2nd") and the sloppiest eigendirection ("last") derived from the CVAE surrogates. For all models except the Mark-0, RMSE(1st) > RMSE(2nd) > RMSE(last), indicating the CVAEs are identifying relatively stiff and sloppy eigendirections. For each model, fifteen different parameter vectors were used, each drawn from the distributions specified in Appendix C.1.

3.4.7 Speed comparison – All Models

So far, we have demonstrated that using a CVAE surrogate leads to improvements in the accuracy of eigenvector computation, for most of the models used in this chapter. However, as Table 3.7 shows, computing Jacobians using CVAEs is slower than using FNNs. This is primarily due to the need to generate multiple runs of the CVAE, and apply backpropagation on each run to derive an average Jacobian (in contrast to FNNs, which require only a single backpropagation pass). As a result, the time advantage of using a surrogate rather than SFD decreases, and in the case of the HH model, the advantage of using a surrogate disappears. However, for the slower, more complex models (SIR, RCK, and Mark-0) a clear speed advantage was still present, and given the observed improvement in eigendirection accuracy resulting from using CVAE surrogates, it appears our generative approach would be worthwhile for a range of complex models. This includes models that take significantly more time to simulate than the RCK and Mark-0, such as that of Poledna et al. [118].

	HH	SIR	BH	RCK	Mark-0
Model Simulation time	0.012	0.067	0.0002	9.8	1.0
CVAE simulation time	0.033	0.034	0.034	0.032	0.038
CVAE gradient samples (z)	50	50	50	50	150
CVAE Jacobian time (single run)	0.972	0.118	0.139	0.174	0.155
CVAE Jacobian time (all z samples)	48.6	5.9	6.1	8.2	18.7
SFD samples	100	100	400	30	75
SFD Jacobian time	10.6	44.7	0.002	1546	1459
Speed-up factor (SFD/FNN)	2.2×10^{-1}	7.6	3.3×10^{-4}	1.9×10^{2}	78

Table 3.7: **Compute time for CVAE and SFD methods**. For all models except HH and BH, CVAE Jacobians could be computed significantly faster than their SFD counterparts, as measured by the "Speed-up factor", equal to SFD Jacobian time / CVAE Jacobian time. SFD Jacobians were calculated using forward-differences. SFD sample sizes were reduced for the RCK and Mark-0 models, to make it computationally feasible to experiment with these models. More latent samples (z) were used to compute the Jacobian for the Mark-0 model, reflecting greater observed variability (i.e., greater stochasticity) in the outputs of its trained CVAE surrogate. Times shown are in seconds.

3.4.8 The Eigenvector-Swapping Problem and Convolution

When CS averages were computed over multiple samples, it was common to see two attributes at the same time: diagonal elements with values significantly less than one (indicating failure of the CVAE method). However, blocks of 2×2 matrices on the diagonal with element values significantly above the expected CS value for two random vectors. A possible explanation for this relates to the misidentification of the relative stiffness of two eigenvectors, due to their eigenvalues appearing to be in the incorrect order. This phenomenon was described in Chapter 1, and was often linked to a pair of eigenvalues being of very similar magnitude – and thus difficult to distinguish when estimates were imperfect.

For a given model, suppose we sample n points in parameter space, and at each point perfectly determine the p eigendirections of the model (with $p >> 2$). However, suppose the CVAE's first and second eigenvalues are inaccurately computed (due to the surrogate being an imperfect emulator of the model), such that 50% of the time the first and second eigenvectors are identified in a swapped order (i.e., $v_1^{\text{SFD}} = v_2^{\text{CVAE}}$, and vice versa). If the eigenvalues were correctly computed, the top-left 2×2 block

of each matrix would equal:

$$\begin{bmatrix} 1 & 0 & \cdots \\ 0 & 1 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}. \tag{3.10}$$

However, when eigenvalues are swapped, the top-left is given by:

$$\begin{bmatrix} 0 & 1 & \cdots \\ 1 & 0 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}. \tag{3.11}$$

Averaging over the n samples, the mean CS matrix has upper-left elements:

$$\begin{bmatrix} 0.5 & 0.5 & \cdots \\ 0.5 & 0.5 & \cdots \\ \vdots & \vdots & \ddots \end{bmatrix}. \tag{3.12}$$

In this case, it would appear that the CVAE method has failed to identify any eigendirections accurately. However, the method did in fact identify the first and second eigenvalues perfectly in all cases, albeit with their order swapped 50% of the time.

When applying the CVAE method, it is useful to recognise when the method has failed to perfectly identify the eigenvectors and their ordering. However, it is also useful to determine whether the method has computed eigenvectors accurately but put them in the wrong order, or completely failed to compute any eigenvectors accurately. To gain insight into which is the case, we can apply a convolution operation to the CS matrices.

Matrix Convolution

To determine the degree to which eigenvector swapping has influenced the CS matrix, we make use of matrix convolutions. By convolving a CS matrix (A) with a small "filter" matrix (also known as a "kernel" matrix), we can convert each element of A into weighted sum of its own value and the values of nearby elements, producing a convolved matrix G. To do this, we choose a small 2×2 filter matrix F and apply it to all 2×2 blocks of A (including blocks which overlap with one another). Suppose A and F are given by:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \quad \text{and} \quad F = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

The convolution $A * F$ is computed by sliding the filter F over matrix A, and computing the sum of the element-wise products. As an example, consider the top-left submatrix of A:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

Applying the filter F to this submatrix yields:

$$\text{Result} = (1 \times a_{11} + 2 \times a_{12} + 3 \times a_{21} + 4 \times a_{22})$$

This value is the top-left element of the output matrix G:

$$G = \begin{bmatrix} (a_{11} + 2a_{12} + 3a_{21} + 4a_{22}) & \cdots \\ \vdots & \ddots \end{bmatrix}.$$

This process is repeated on each submatrix of A, yielding the matrix:

$$G = \begin{bmatrix} (a_{11} + 2a_{12} + 3a_{21} + 4a_{22}) & (a_{12} + 2a_{13} + 3a_{22} + 4a_{23}) & \cdots \\ (a_{21} + 2a_{22} + 3a_{31} + 4a_{32}) & (a_{22} + 2a_{23} + 3a_{32} + 4a_{33}) & \\ \vdots & & \ddots \end{bmatrix}.$$

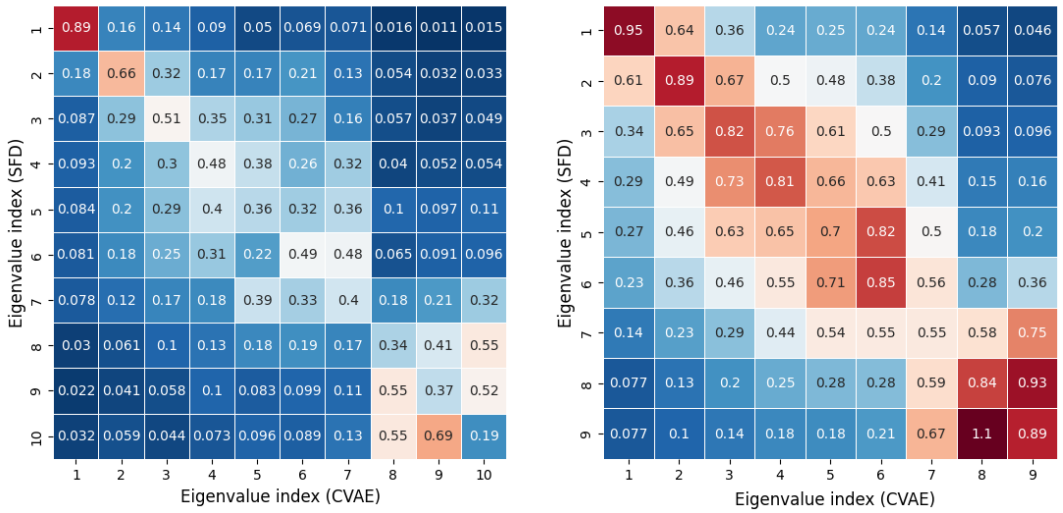
Convolution Result (Mark-0)

The filter chosen for convolving was:

$$F = \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}. \tag{3.13}$$

The elements of F were chosen such that perfectly diagonal CS matrices (corresponding to perfect eigenvector alignment and ordering) will maintain a value of unity for each element of the diagonal when convolved with F. As the filter is a 2×2 matrix, the convolved form of a $p \times p$ matrix will be of dimension $(p-1) \times (p-1)$.

The result of convolving the CS similarity matrix for the Mark-0 using F is shown in Figure 3.8, along with the original CS matrix.



CS matrix **Convolved CS matrix**

Table 3.8: **CS matrix and convolved CS matrix for Mark-0**. The diagonal of the convolved matrix has values much closer to unity, compared to the original CS matrix. This indicates that eigenvector pairs were "swapped" in many of our test points in parameter space, leading to lower CS averages, even where eigendirections were accurately identified. A 2×2 filter was used. Parameter vectors were the same as those used in Figure 3.5.

What could be described as a "spreading tail" (the bottom-right section of the diagonal) was observed in the convolved matrix for the Mark-0 (Figure 3.8). This indicates that the low cosine similarity of the sloppy CVAE and SFD directions was not simply a result of swapping pairs of consecutive eigenvectors. Instead, either (a) eigenvector swapping often took place between non-consecutive vectors (e.g. the 9th and 6th vectors), or (b) each sloppy CVAE eigenvector was a linear combination of several sloppy SFD eigenvectors. If neither of these were the case, we would not expect the "wings" of the convolved CS matrix (the top-right and bottom-left corners) to have much smaller elements than those of the tail (which is indicated by the blue shading of the wings).

The higher values observed on the diagonal of the convolved CS matrix indicate that eigenvectors were approximately identified, even if they were often switched or linearly mixed. This is arguably a promising sign, as it shows our (suboptimal) CVAEs are capable of discovering sloppy eigenvectors, to some extent. It appears likely that more accurate CVAEs would be even more capable of identifying sloppy eigendirections.

3.5 Discussion

The CVAE surrogates used in this chapter showed greater ability to replicate the behaviour of the most complex stochastic models tested, compared to the deterministic surrogates used in the previous chapter. Clearly, the use of generative ANNs as surrogates for nondeterministic models is a worthwhile line of research, given: (a) The method's speed advantage compared to SFD when computing the eigendirections of the models (even if this is only present in slower and more complex models), and (b) the ability of generative surrogates to better capture the inherent stochasticity present in many models, compared to deterministic surrogates. However, for models where the deterministic surrogate method excelled, such as the Agent-based SIR, little or no gain was realised. It appears that when a model has a limited degree of stochasticity – small enough for a deterministic surrogate to sufficiently emulate its sensitivities – a generative surrogate is unnecessary.

It seems very likely that the neural network methods used here are suboptimal, in the sense that substantial improvement in predictive power may be achievable using alternative CVAE architectures and training techniques. For example, convolutional or recurrent structures within the models may be able to capture the highly autocorrelated "shapes" of time series curves more realistically than the internal FNN layers we have used inside our CVAEs. Additionally, conditional vector-quantised VAEs (VQ-VAEs) may further increase performance, particularly in cases where the target model shows distinct "modes" of behaviour (such as those of the HH model), with rapid transitions between these modes occurring as one moves through parameter space [98]. However, the capability of simple CVAE structures to model a wide range of models – as we have observed – is useful, as it means users of these methods may not need to test a wide range of CVAE network structures (including layer count, size, and use of FNNs versus convolutional networks within the autoencoders) to achieve a reasonable level of identification of the stiff eigenvectors.

Generative networks other than autoencoders may also be effective, such as synthetic neural likelihood methods. These methods can compute an approximation of a model's likelihood function [119], from which a FIM estimate can be calculated. A collaboration with Dr Joel Dyer indicated this method may be of value, though no clear performance gains were observed over the CVAE technique. However, further research into SNL seems justified, given the potential advantages of estimating a FIM directly rather than deriving it from full time series outputs.

3.5.1 Predicting the need for Nondeterministic Surrogates

In our results, it was notable that for the HH model and the agent-based SIR, CVAE surrogates provided little or no advantage over FNNs. When deciding whether to use deterministic networks or generative ones as surrogates for a nondeterministic model, it is useful to consider the form of the nondeterminism in the model of interest. If the stochasticity in model outputs is relatively small, and changes in the output time series (e.g., its location and shape) are dominated by changes in the parameters, then the "averaging" behaviour of a deterministic surrogate may not be problematic. An FNN surrogate – which has learned the model's average behaviour – will be able to adjust its location and shape appropriately when its input parameters are varied, as was the case for the SIR ABM. Conversely, if oscillations in a model's output are large, or if its parameters have minimal effect on the average output (and instead affect the phase or magnitude of oscillations, for example), then it is likely that CVAE surrogates will better capture the model's parameter sensitivity, and are likely a more suitable starting point.

3.5.2 Beyond Sloppiness: Generative Surrogates for Efficient Analysis and Decision Support

In this thesis, our proposed application of surrogate ANNs has been centred on computing stiff and sloppy eigendirections. Whether our surrogates have been able to replicate complex behaviours (primarily fluctuations in outputs) has been secondary to the goal of mimicking parameter sensitivities. However, if a surrogate is capable of mimicking both the behaviours and sensitivities of models, another application is possible. By using CVAEs and other generative networks to act as fast emulators for slow stochastic models, it would be possible to quickly assess the impact of a known set of possible policies and interventions, without having to travel incrementally through eigendirections to explore parameter space. For example, in an economic ABM, the impact of a policy (or a set of possible policies) could be done quickly via a generative surrogate, and where the most desirable outcome is determined, the ABM itself can be run to validate this result. Similarly, if one is seeking to optimise the risk-reward trade-off of a financial strategy (reward being the mean path, and risk corresponding to fluctuations), one could use a generative surrogate to quickly investigate many possible investment choices, in place of a slow but reliable model. Once an optimal portfolio is chosen using the surrogate, validation (using the slower model) could be carried out to ensure the investment strategy is robust.

3.6 Chapter 3 Conclusion

In this chapter, we described generative neural network models and discussed their history and applications. We then described our method of creating nondeterministic surrogates, using a type of generative network – a CVAE. Their ability to replicate the behaviour of nondeterministic models was evaluated, and their performance in eigenvector identification was found to be generally superior to that of the deterministic surrogates used in Chapter 2, particularly when used on ABMs. The widespread applicability of this method is notable. A wide range of baseline models were used as test cases, and for all models, a significant degree of success was shown in identifying the stiff eigendirections in particular. This included models where deterministic surrogates also performed well.

The results of this chapter justify further research into the use of generative surrogates for identifying stiff and sloppy eigendirections. The performance advantages observed over deterministic surrogates – and the conceptual justification for why a generative surrogate can better model highly variant constant-mean systems – suggests further research into deterministic surrogates would be less promising than studying how to improve generative surrogation methods.

Thesis Conclusion

Variability and uncertainty are widely observed in economic systems around the world. This has led to the creation of economic models incorporating nondeterminism in numerous ways. There is ongoing debate about the suitability of different approaches to building nondeterministic economic models – including the presence of exogenous shocks and assumptions of agent rationality – and many points of contention remain after decades of debate. Whichever approach a modeller takes, it is valuable to gain insight into the workings of the model of interest, including the different behaviours it is capable of exhibiting, and its sensitivity to parameter variation. The concept of parameter sloppiness has been used to gain such insights, in economics and other fields.

The primary contributions of this thesis are three novel methods for efficiently computing stiff and sloppy eigendirections in stochastic economic models. The methods developed are applicable to multiple classes of economic models, including DSGEs (via our IRF method) and ABMs (via our surrogate methods). The two surrogate methods can also be used with non-economic models, making them applicable to fields beyond economics. Our first method involved the use of IRFs, and was applicable to DSGE economic models and VARs. This method successfully identified stiff directions across a range of DSGEs, but was less accurate in identifying sloppy directions. Testing this method's applicability to other models (e.g., vector error correction models and other modern statistical models used in economics) seems worthwhile. However, the requirement for models to have meaningful "impulse responses" limits the method's applicability beyond DSGEs and VARs.

The IRF method provides a capability to explore the significance – and insignificance – of different parameters, providing insight into the behaviour of DSGE models. Because economic systems frequently undergo switching between distinct behavioural regimes (including booms, busts, and crises) it is crucial for researchers to confirm their models are able to capture these dynamics. Additionally, guidance as to which

parameters and parameter combinations most influence the state of the economy is useful for policy analysts who wish to find and implement impactful economic policies.

The second method we developed was to use conventional sloppiness analysis on model surrogates, trained to mimic the behaviour of the economic models of interest. The surrogates of Chapter 2 took the form of deterministic FNNs, built to act as emulators for the economic models. Computing the derivatives of these surrogates via the backpropagation algorithm was fast, providing an efficient way to compute eigendirections for several models studied. Stiff eigendirections were more accurately computed than sloppy ones, making this method more suitable for parameter space exploration rather than model reduction (using MBAM or other techniques).

Our third method used generative surrogate models, in the form of CVAEs. CVAEs were clearly superior to deterministic FNNs, and were able to solve the "zero-mean" problem which deterministic surrogates were unable to handle. Although they required multiple backpropagation runs to compute derivatives, computing eigendirections using these surrogates was still far faster than using the models themselves. Additionally, models with a high degree of stochasticity were better emulated by the CVAE surrogates than they were by FNNs. The ability to apply our surrogate methods to a wide range of models underscores their flexibility and potential in economic modelling and other fields.

This thesis provides multiple avenues for future research. Direct improvements on the methods developed in Chapters 2 and 3 are likely possible, as the surrogates developed in this thesis are likely suboptimal in terms or performance as emulators. This suggests investigating the use of more sophisticated deterministic neural network architectures may significantly improve performance. While it is likely that significant improvements can be made to our deterministic surrogates, the superior performance of our generative surrogates suggests further development of these is more worthwhile. The ability of CVAE models (and other generative ANNs) to emulate the stochasticity inherent in nondeterministic economic models makes them particularly suitable for further research. In addition to further developing our generative surrogate method, another possible line of investigation is the use of summarising surrogates. These could involve computing the FIM from the latent nodes of an autoencoder network, or using synthetic neural likelihood methods.

As the argument for modelling the economy as a complex system has become prominent, the development of ABMs and other unconventional models has accelerated. Many of these models show emergent behaviours which cannot be easily

predicted from the models' rules, and may only occur in certain regions of parameter space. Studying the behaviours of these new models is crucial, and this thesis contributes new methods for doing so.

Appendix A

A.1 FIM for a Multivariate Gaussian Distribution

The probability density function of a multivariate Gaussian distribution with mean vector $\boldsymbol{\mu}(\theta)$ and covariance matrix Σ is:

$$p(\mathbf{x};\theta) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}(\theta))^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}(\theta))\right). \tag{A.1}$$

The log-likelihood function is thus:

$$\ell(\theta) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\Sigma| - \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}(\theta))^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}(\theta)). \tag{A.2}$$

Its first derivative is:

$$\frac{\partial \ell(\theta)}{\partial \theta_i} = \frac{\partial}{\partial \theta_i}\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}(\theta))^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}(\theta))\right) \tag{A.3}$$

$$\frac{\partial \ell(\theta)}{\partial \theta_i} = (\mathbf{x}-\boldsymbol{\mu}(\theta))^\top \Sigma^{-1}\frac{\partial \boldsymbol{\mu}(\theta)}{\partial \theta_i} \tag{A.4}$$

Using the chain rule, this simplifies to:

$$\frac{\partial \ell(\theta)}{\partial \theta_i} = (\mathbf{x}-\boldsymbol{\mu}(\theta))^\top \Sigma^{-1}\frac{\partial \boldsymbol{\mu}(\theta)}{\partial \theta_i} \tag{A.5}$$

$$= \frac{\partial \boldsymbol{\mu}(\theta)}{\partial \theta_i}^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}(\theta)). \tag{A.6}$$

The Hessian of the log-likelihood is therefore:

$$\frac{\partial^2 \ell(\theta)}{\partial \theta_i \partial \theta_j} = -\frac{\partial \boldsymbol{\mu}(\theta)}{\partial \theta_j}^\top \Sigma^{-1}\frac{\partial \boldsymbol{\mu}(\theta)}{\partial \theta_i} + \frac{\partial^2 \boldsymbol{\mu}(\theta)}{\partial \theta_i \partial \theta_j}^\top \Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}(\theta)). \tag{A.7}$$

Let this be called H_{ij}. The FIM is defined as the expected value of the curvature of the log-likelihood, i.e. the expected value of H. Over the distribution of possible $\mathbf{x}$ values, we have

$$E[\mathbf{x}|\theta] = \boldsymbol{\mu}(\theta), \tag{A.8}$$

$$E[\mathbf{x} - \boldsymbol{\mu}(\theta)|\theta] = 0, \tag{A.9}$$

meaning the second term of H vanishes. We are left with:

$$I_{ij}(\theta) = -\frac{\partial \boldsymbol{\mu}(\theta)}{\partial \theta_j}^{\top} \Sigma^{-1} \frac{\partial \boldsymbol{\mu}(\theta)}{\partial \theta_i} \tag{A.10}$$

which may be written

$$I_{ij}(\theta) = \sum_{m=1}^{p} \sum_{n=1}^{p} \frac{\partial \mu_{,}m(\theta)}{\partial \theta_i} \left(\Sigma^{-1}\right)_{mn} \frac{\partial \mu_n(\theta)}{\partial \theta_j}. \tag{A.11}$$

If Σ is diagonal, then:

$$\Sigma = \begin{bmatrix} \sigma_1^2 & 0 & \cdots & 0 \\ 0 & \sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_K^2 \end{bmatrix} \tag{A.12}$$

and

$$\Sigma^{-1} = \begin{bmatrix} 1/\sigma_1^2 & 0 & \cdots & 0 \\ 0 & 1/\sigma_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1/\sigma_K^2 \end{bmatrix}. \tag{A.13}$$

Therefore:

$$I_{ij}(\theta) = \sum_{k=1}^{K} \frac{1}{\sigma_k^2} \frac{\partial \mu_k(\theta)}{\partial \theta_i} \frac{\partial \mu_k(\theta)}{\partial \theta_j}. \tag{A.14}$$

A.2 KL Divergence

KL divergence is an important concept in statistics, with applications in machine learning, information theory, and sloppiness analysis. The KL divergence between two discrete probability distributions P and Q is defined as:

$$D_{KL}(P\|Q) = \sum_{x \in X} P(x) \log\left(\frac{P(x)}{Q(x)}\right). \tag{A.15}$$

For continuous distributions, it takes the form:

$$D_{KL}(P\|Q) = \int_{-\infty}^{\infty} P(x) \log\left(\frac{P(x)}{Q(x)}\right) dx. \tag{A.16}$$

Features of KL divergence include:

1. It is asymmetric, i.e. $D_{KL}(P\|Q) \neq D_{KL}(Q\|P)$, in general.

2. It does not satisfy the triangle inequality, and is therefore not a true metric.

3. If $Q(x) = 0$ for any x with non-zero probability under P (i.e. $Q(x) = 0$, but $P(x) > 0$), then the $D_{KL}(P\|Q)$ will be infinite.

    - Disjoint probability distributions will therefore have infinite KL divergence.

KL divergence can be useful as a measure of distance between distributions, and is mathematically linked to a number of concepts, including model sloppiness.

Appendix B

B.1 DSGE Model Details

In this section, we list the parameters of the DSGE models we have used, along with a brief description for each. We also provide the parameter values used in when we investigated sloppiness in these models. We do not describe the model equations in detail, as these DSGE models can consist of dozens of equations.

B.1.1 Smets & Wouters model

The SW model is built upon assumptions and mechanism which are common amongst modern DSGE models. It is microfounded, based on agents who use rational expectations to solve utility-optimisation problems. The model contains nominal rigidities, which slow the rate at which the model tends towards equilibrium following an exogenous shock. The parameters varied during sloppiness analysis, and the parameters held constant are listed below.

SW Varied Parameters

Parameter	Value	Meaning
δ	0.025	Depreciation rate
ϕ_w	1.5	Gross markup wages
$\bar{g}/\bar{y}$	0.0324	Steady state exogenous spending share
ε_p	10	Curvature Kimball aggregator for prices
ε_w	10	Curvature Kimball aggregator for wages

Table B.1: Parameter values and their meanings.

SW Fixed Parameters

Parameter	Value	Meaning
ρ_{ga}	0.2601	Feedback technology on exogenous spending
l	0	Steady state hours
$\bar{\pi}$	0.7	Steady state inflation rate
$100(\beta^{-1}-1)$	0.742	Time preference rate in percent
μ_w	0	Coefficient on MA term wage markup
μ_p	0	Coefficient on MA term price markup
α	0.0576	Capital share
ψ	0.2696	Capacity utilisation cost
φ	6.0144	Investment adjustment cost
σ_c	1.5	Risk aversion
λ	0.40462321	External habit degree
ϕ_p	2.25	Fixed cost share
ι_w	0.3243	Indexation to past wages
ξ_w	0.8087	Calvo parameter wages
ι_p	0.47	Indexation to past prices
ξ_p	0.6	Calvo parameter prices
σ_l	1.9423	Frisch elasticity
r_π	1.488	Taylor rule inflation feedback
$r_{\Delta y}$	0.2347	Taylor rule output growth feedback
r_y	0.0593	Taylor rule output level feedback
ρ	0.8762	Interest rate persistence
ρ_a	0.9977	Persistence productivity shock
ρ_b	0.5799	Persistence risk premium shock
ρ_g	0.9957	Persistence spending shock
ρ_i	0.7165	Persistence risk premium shock
ρ_r	0	Persistence monetary policy shock
ρ_p	0	Persistence price markup shock
ρ_w	0	Persistence wage markup shock
$\bar{\gamma}$	0.3982	Net growth rate in percent

Table B.2: Parameter values and their meanings.

SW Shocks

Shock Variable	Variance	Meaning
η^a	0.4618	Productivity shock
η^b	1.8513	Risk premium shock
η^g	0.6090	Spending shock
η^i	0.6017	Investment-specific technology shock
η^m	0.2397	Monetary policy shock
η^p	0.1455	Price markup shock
η^w	0.2089	Wage markup shock

Table B.3: Shock parameters and their meanings.

B.1.2 Aguiar & Gopinath VAR model

Aguiar & Gopinath Varied Parameters

Parameter	Value	Meaning
β	1/1.02	Time preference rate
γ	0.36	Consumption exponent (utility)
b	0.1	Steady-state debt to GDP ratio
ψ	0.001	Coefficient on interest rate premium
α	0.68	Labour exponent (production)

Table B.4: Parameter values and their meanings.

Aguiar & Gopinath Fixed Parameters

Parameter	Value	Meaning
σ	2	Risk Aversion
δ	0.05	Depreciation Rate
ϕ	4	Capital adjustment cost parameter
μ_g	log(1.0066)	Mean Growth Rate
ρ_z	0.95	Autocorrelation of technology coefficient
ρ_g	0.01	Autocorrelation of government spending

Table B.5: Parameter values and their meanings.

Aguiar & Gopinath Shocks

Shock Variable	Variance	Meaning
ε_g	0.0281	Government spending shock
ε_z	0.0048	Technology shock

Table B.6: Shock parameters and their meanings.

The paper provided two values for each shock variance ($\sigma_g^2 \in \{0.0281, 0.0088\}$ and $\sigma_z^2 \in \{0.0088, 0.0078\}$). The default values from *DSGE_mod* were used, which were calculated previously by fitting the model to Mexican economic data (the alternative being the Canadian data).

B.1.3 Gali 2015 (Chapter 8) model

Gali 2015 (Chapter 8) Varied Parameters

Parameter	Value	Meaning
β	0.99	Discount factor
σ	1	Inverse elasticity of intertemporal substitution
φ	5	Inverse Frisch elasticity
α	0.25	Capital share
ϵ	9	Demand elasticity

Table B.7: Parameter values and their meanings.

Gali 2015 (Chapter 8) Fixed Parameters

Parameter	Value	Meaning
θ	0.75	Calvo parameter
υ	0.4	Openness parameter
η	1	Substitutability of foreign and domestic goods
ρ_a	0.9	Autocorrelation technology shock
ρ_ν	0.5	Autocorrelation monetary policy shock
ρ_{y^*}	0	Autocorrelation world output growth shock
ρ_z	0.5	Autocorrelation preference shock
ϕ_π	1.5	Inflation feedback Taylor Rule
ϕ_y	0.125	Output feedback Taylor Rule

Table B.8: Parameter values and their meanings.

Gali 2015 (Chapter 8) Shocks

Shock Variable	Variance	Meaning
ε^{ν}	0.25^2	Monetary policy shock
ε^{a}	0.25^2	Technology shock
ε^{z}	0.25^2	Preference shock

Table B.9: Shock parameters and their meanings.

The default values for all shocks are zero, except for ε_a.

B.1.4 Schmitt-Grohe & Uribe model

The SGU model used in this thesis is the "Neoclassical Growth Model" (Example 1, section 5.1) variant from Schmitt-Grohe & Uribe 2004 [126].

SGU Varied Parameters

Parameter	Value	Meaning
β	0.95	Utility discount rate
δ	1	Capital depreciation rate
α	0.3	Capital share of income
ρ	0	Autocorrelation of technology coefficient
σ	2	Intertemporal elasticity of substitution

Table B.10: Parameter values and their meanings.

All parameters of the SGU model were varied when computing model sloppiness – no parameters were fixed.

SGU Shock

Shock Variable	Variance	Meaning
σ_A	1	Technology shock

Table B.11: Shock parameter and its meaning.

B.1.5 An RBC model

RBC Varied Parameters

Parameter	Value	Meaning
σ	1	Risk aversion
α	0.33	Capital share
I/Y	0.25	Investment-output ratio
K/Y	10.4	Capital-output ratio

Table B.12: Parameter values and their meanings.

RBC Fixed Parameters

Parameter	Value	Meaning
ρ_z	0.97	Persistence of technology shock
ρ_g	0.989	Persistence of government spending shock
G/Y	0.2038	Government spending share
n	0.0027	Population growth
x	0.0055	Technology growth (per capita output growth)
$\bar{G}$	0.2038	Government spending in steady state

Table B.13: Parameter values and their meanings.

RBC Shocks

Shock Variable	Variance	Meaning
ε_z	0.4356	Technology shock
ε_g	1.0816	Government spending shock

Table B.14: Shock parameters and their meanings.

B.1.6 Hansen model

Hansen Varied Parameters

Parameter	Value	Meaning
β	0.99	Utility discount rate
δ	0.025	Capital depreciation rate
θ	0.36	Capital share of income
γ	0.95	Autocorrelation of technology coefficient
A	2	Preference for leisure against consumption

Table B.15: Parameter values and their meanings.

Hansen Fixed Parameter

Parameter	Value	Meaning
h_0	0.53	Total number of workers (scaled)

Table B.16: Parameter values and their meanings.

Hansen Shock

Shock Variable	Variance	Meaning
ε_a	0.00712	Technology shock

Table B.17: Shock parameters and their meanings.

Appendix C

C.1 Model Parameters, Outputs, and Initial Conditions

Below, we list the parameter ranges used for each model. Parameters were drawn from uniform distributions over these ranges when generating sample runs, training sets, and the parameter vectors used as sample points for calculating cosine similarity matrices. Initial conditions and output normalisation formulae (used to ensure the models' outputs all lie within $[0, 1]$) are also listed.

C.1.1 Toy model $\phi(\theta, t)$

The output was normalised for ANN training using the following transformation:

$$Y(\theta, t) = \frac{1}{3}\phi(\theta, t). \tag{C.1}$$

The step size for computing SFD Jacobians was 8% of each parameter's range (i.e., to compute $\partial Y_i/\partial\theta_0$, the parameter θ_0 was perturbed by 8% of the range listed in Table C.1).

Parameter type	**Parameter**	**Value or Range**	**Description**
Varied	θ_0	[0.2, 0.8]	N/A
Varied	θ_1	[0.2, 0.8]	N/A
Varied	θ_2	[0.2, 0.8]	N/A
Varied	θ_3	[0.2, 0.8]	N/A
Fixed	σ	0.02	Standard deviation of shock

Table C.1: Toy model $\phi(\theta, t)$ parameters ranges and meanings.

C.1.2 Hodgkin-Huxley Model

The output used was four time series normalised and concatenated into a single univariate time series $(\frac{1}{81}V(t), m(t), h(t), n(t))$. The initial conditions were:

$$V(0) = -65 + \varepsilon_{V,0} \tag{C.2}$$

$$m(0) = 0.05 + \varepsilon_{m,0} \tag{C.3}$$

$$h(0) = 0.6 + \varepsilon_{h,0} \tag{C.4}$$

$$n(0) = 0.32 + \varepsilon_{n,0}, \tag{C.5}$$

where the ε values are the realisations of the additive Gaussian noise $t = 0$. The step size for computing SFD Jacobians was 8% of each parameter's range.

Parameter type	**Parameter**	**Value or Range**	**Description**
Varied	C_m	[0.7, 1.3]	Capacitance of the lipid bilayer
Varied	g_{Na}	[84, 156]	Sodium channel maximum conductance
Varied	g_K	[25.2, 46.8]	Potassium channel maximum conductance
Varied	g_L	[0.21, 0.39]	Leak channel maximum conductance
Varied	E_{Na}	[35, 65]	Sodium source voltage
Varied	E_K	[-100.1, -53.9]	Potassium source voltage
Varied	E_L	[-70.7, -38.1]	Leak source voltage
Fixed	I_{ext}	10	External current
Fixed	σ	0.02	Standard deviation of additive Gaussian noise. (Same value for all output variables.)

Table C.2: HH parameters ranges and meanings.

C.1.3 Agent-based SIR

The output used was the population infection rate. The only initial condition was the number of people infected at $t = 0$, which is $N \times \lambda_0$ (values given in table below). The step size for computing SFD Jacobians was 8% of each parameter's range.

Parameter type	Parameter	Value or Range	Description
Varied	p_I	[0.174, 0.426]	Infection probability after interaction with infected person
Varied	p_R	[0.058, 0.142]	Probability of recovering from infection, per unit time
Varied	λ_0	[0.058, 0.142]	Proportion of population initially infected
Varied	β	[0.29, 0.71]	Rewiring probability of Watts–Strogatz network
Fixed	N	1000	Population
Fixed	K	2	Mean degree of network

Table C.3: Agent-based SIR parameter ranges and meanings.

C.1.4 Brock-Hommes

The output variable used was the asset price, $p(t)$. This did not require scaling. The initial condition was $p(0) = 0$. The step size for computing SFD Jacobians was 8% of each parameter's range.

Parameter type	Parameter	Value or Range	Description
Varied	g_1	[0.2, 0.8]	Trend sensitivity, strategy 1
Varied	b_1	[0.2, 0.8]	Forecast bias, strategy 1
Varied	g_2	[0.2, 0.8]	Trend sensitivity, strategy 2
Varied	b_2	[0.2, 0.8]	Forecast bias, strategy 2
Fixed	β	10	Intensity of choice
Fixed	σ	0.04	Standard deviation of price shocks
Fixed	r	0.01	Risk-free interest rate
Fixed	g_3	1.01	Trend sensitivity, strategy 3
Fixed	b_4	0	Forecast bias, strategy 3
Fixed	p^*	0	Equilibrium price level

Table C.4: BH parameter ranges and meanings.

C.1.5 RCK ABM

The output variable was economy-wide capital, normalised for training using the following transformation:

$$Y(\theta, t) = \log(1 + \text{Consumption}(\theta, t)/10^8). \tag{C.6}$$

The logarithmic transformation was used to shrink the large endogenous fluctuations in the RCK model, which ensured the order of magnitude of the output did not vary

as greatly as it does in the original model. As described in Section 2.6.5, the first 500 time periods were ignored, to remove an erratic transient period occurring at the beginning of simulations. Initial capital was one unit per capita, and the households' initial savings rates were drawn independently from a uniform distribution, $U(0, 1)$. The step size for computing SFD Jacobians was 8% of each parameter's range.

Parameter type	**Parameter**	**Value or Range**	**Description**
Varied	α	[0.56, 0.74]	Capital elasticity
Varied	d	[0.07, 0.13]	Depreciation rate
Varied	ϕ	[0.008, 0.017]	Labour normed standard deviation
Varied	τ	[24, 36]	Mean savings rate update interval
Fixed	N	500	Population

Table C.5: RCK ABM parameter ranges and meanings.

C.1.6 Mark 0

The output variable used was the unemployment rate, which did not require scaling. The initial state was derived from the parameter values (i.e., no initial state had to be chosen, as parameter values determined the initial state). The step size for computing SFD Jacobians was 8% of each parameter's range.

Parameter type	Parameter	Value or Range	Description
Varied	R	[0.8, 3.2]	Hiring/firing rate
Varied	α_γ	[20, 80]	Loan rate effect on R
Varied	Γ_0	[0.02, 0.08]	Baseline gamma
Varied	r	[0.4, 1.6]	Adjustment ratio γ_w/γ_p
Varied	γ_p	[0.02, 0.08]	Price-adjustment size
Varied	η_0	[0.04, 0.16]	Baseline firing propensity
Varied	W_f	[0.4, 1.6]	Wage adjustment to inflation
Varied	$\rho^\star$	[0.032, 0.128]	Baseline interest rate
Varied	ϕ_π	[0, 0.001]	CB reaction to inflation
Varied	ϕ_ε	[0, 0.001]	CB reaction to employment
Fixed	$\pi^\star$	0.002	CB inflation target
Fixed	$\hat{\varepsilon}^\star$	0.05	CB unemployment target
Fixed	Θ	5	Loan default threshold
Fixed	f	1	Bankruptcy effect on bank interest rates
Fixed	α_c	4	Real rate influence on consumption
Fixed	c_0	0.5	Baseline propensity to consume
Fixed	δ	0.02	Dividend share
Fixed	β	0	Household intensity of choice
Fixed	ω	0.2	Exponentially weighted moving average memory
Fixed	τ^R	0.5	Weight of π^{ema} on $\hat{\pi}$
Fixed	τ^T	0.5	Weight of $\pi^\star$ on $\hat{\pi}$
Fixed	ϕ	0.1	Revival frequency
Fixed	y_0	0.5	Initial production
Fixed	N	5000	Population

Table C.6: Mark 0 parameter ranges and meanings.

C.2 Deterministic Surrogate Network Specifications

The specifications for each FNN – including number of layers, layer size, and activation functions – are listed in Table C.7. Figure C.1 is an illustration of the CVAE surrogate structure used for the agent-based SIR model. After experimenting with possible FNN structures for the HH model's deterministic surrogate, we settled on an FNN with four hidden layers, and ReLU activations. For the subsequent models (ϕ, SIR, BH, RCK, Mark-0), the same four-layer network was used as a starting point, with the number of layers and layer sizes varied, in the search for possible performance improvements.

No significant performance improvements were found, so this structure was used for all models.

Model	Hidden Layers	Layer Sizes	Activations	Epochs
$\phi(\theta, t)$	4	16, 64, 128, 128	ReLU	150
HH	4	16, 64, 128, 128	ReLU	400
SIR	4	16, 64, 128, 128	ReLU	100
BH	4	16, 64, 128, 128	ReLU	150
RCK	4	16, 64, 128, 128	ReLU	300
Mark-0	4	16, 64, 128, 128	ReLU	350

Table C.7: FNN specifications for each model. A mean square error loss function was used, with an Adam optimiser. The initial learning rate was 10^{-3}, and batch sizes were 200.

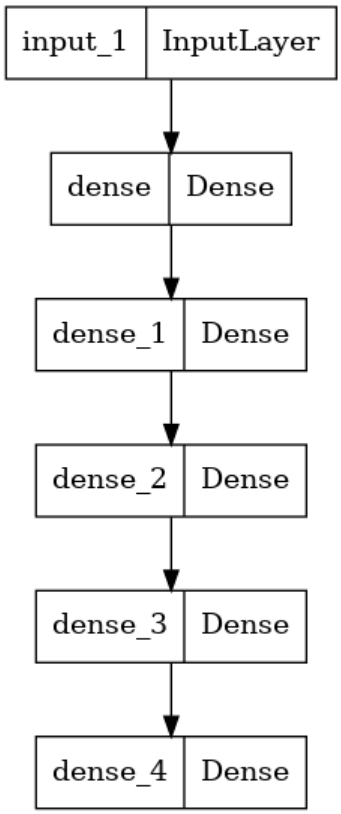


Figure C.1: **FNN Structure** of the deterministic surrogate used for the agent-based SIR model. The Keras software library was used to generate the illustration.

Appendix D

D.1 Generative Surrogate Network Specifications

The specifications for each CVAE network – including layer sizes and activation functions – are listed in the tables below. Figure D.1 is an illustration of the CVAE surrogate structure used for the agent-based SIR model. The same basic structure was used for each CVAE network. However, a deeper network (with more layers) was used for the RCK surrogate, and an additional latent node was added for the Mark-0 surrogate. The performance gained by making these changes likely reflects the higher complexity of the RCK and Mark-0, compared to the other models (for which adding additional hidden layers and latent nodes did not improve performance).

D.1.1 Hodgkin-Huxley Model

Component	Layer Type	Layer Size	Activation Function
Encoder	Dense	128	ReLU
Encoder	Dense	128	Leaky ReLU
Encoder	Dense	64	Leaky ReLU
Encoder	Dense	64	Leaky ReLU
Encoder	Dense	32	Leaky ReLU
Latent	Sampling	2	N/A
Decoder	Dense	32	Leaky ReLU
Decoder	Dense	64	Leaky ReLU
Decoder	Dense	64	Leaky ReLU
Decoder	Dense	128	Leaky ReLU
Decoder	Dense	128	ReLU

Table D.1: HH CVAE Network Specifications. A mean square error loss function was used, with an Adam optimiser. The initial learning rate was 10^{-3}, and batch sizes were 200.

D.1.2 Agent-based SIR

Component	Layer Type	Layer Size	Activation Function
Encoder	Dense	128	ReLU
Encoder	Dense	128	Leaky ReLU
Encoder	Dense	64	Leaky ReLU
Encoder	Dense	64	Leaky ReLU
Encoder	Dense	32	Leaky ReLU
Latent	Sampling	2	N/A
Decoder	Dense	32	Leaky ReLU
Decoder	Dense	64	Leaky ReLU
Decoder	Dense	64	Leaky ReLU
Decoder	Dense	128	Leaky ReLU
Decoder	Dense	128	ReLU

Table D.2: Agent-based SIR CVAE Network Specifications. A mean square error loss function was used, with an Adam optimiser. The initial learning rate was 10^{-3}, and batch sizes were 200.

D.1.3 Brock-Hommes

Component	Layer Type	Layer Size	Activation Function
Encoder	Dense	128	ReLU
Encoder	Dense	128	Leaky ReLU
Encoder	Dense	64	Leaky ReLU
Encoder	Dense	64	Leaky ReLU
Encoder	Dense	32	Leaky ReLU
Latent	Sampling	2	N/A
Decoder	Dense	32	Leaky ReLU
Decoder	Dense	64	Leaky ReLU
Decoder	Dense	64	Leaky ReLU
Decoder	Dense	128	Leaky ReLU
Decoder	Dense	128	ReLU

Table D.3: Brock-Hommes CVAE Network Specifications. A mean square error loss function was used, with an Adam optimiser. The initial learning rate was 10^{-3}, and batch sizes were 200.

D.1.4 RCK ABM

Component	Layer Type	Layer Size	Activation Function
Encoder	Dense	128	ReLU
Encoder	Dense	128	Leaky ReLU
Encoder	Dense	64	Leaky ReLU
Encoder	Dense	64	Leaky ReLU
Encoder	Dense	64	Leaky ReLU
Encoder	Dense	64	Leaky ReLU
Encoder	Dense	32	Leaky ReLU
Latent	Sampling	2	N/A
Decoder	Dense	32	Leaky ReLU
Decoder	Dense	64	Leaky ReLU
Decoder	Dense	64	Leaky ReLU
Decoder	Dense	64	Leaky ReLU
Decoder	Dense	64	Leaky ReLU
Decoder	Dense	128	Leaky ReLU
Decoder	Dense	128	ReLU

Table D.4: RCK ABM CVAE Network Specifications. A mean square error loss function was used, with an Adam optimiser. The initial learning rate was 10^{-3}, and batch sizes were 200.

D.1.5 Mark 0

Component	Layer Type	Layer Size	Activation Function
Encoder	Dense	128	ReLU
Encoder	Dense	128	Leaky ReLU
Encoder	Dense	64	Leaky ReLU
Encoder	Dense	64	Leaky ReLU
Encoder	Dense	32	Leaky ReLU
Latent	Sampling	3	N/A
Decoder	Dense	32	Leaky ReLU
Decoder	Dense	64	Leaky ReLU
Decoder	Dense	64	Leaky ReLU
Decoder	Dense	128	Leaky ReLU
Decoder	Dense	128	ReLU

Table D.5: Mark 0 CVAE Network Specifications. A mean square error loss function was used, with an Adam optimiser. The initial learning rate was 10^{-3}, and batch sizes were 200.

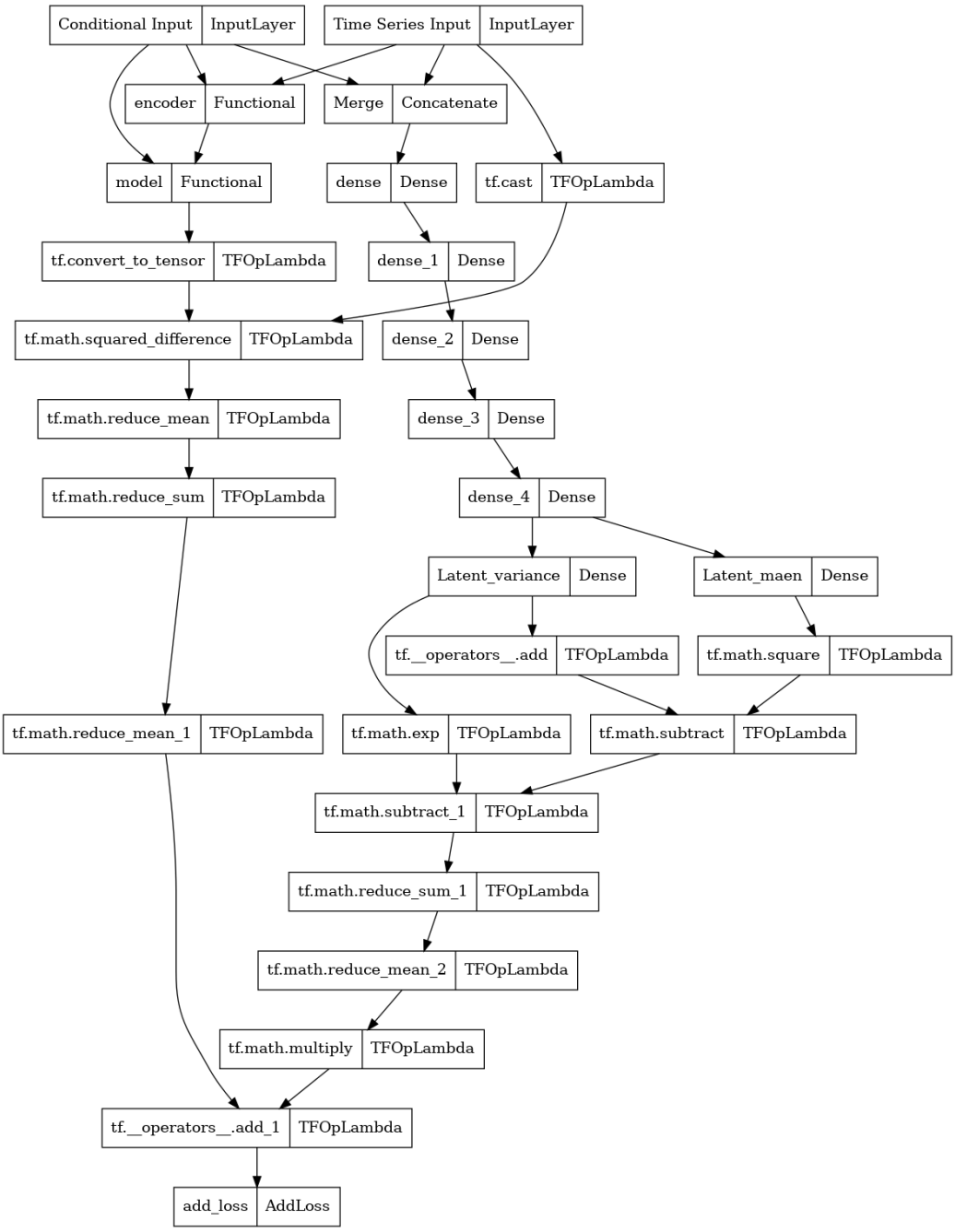


Figure D.1: **Full CVAE Structure** of the generative surrogate used for the agent-based SIR model. The encoder/decoder networks are shown on the right side of the figure, and the left shows the conditional input (θ) flowing into both the beginning of the encoder, and the decoder network after the bottleneck. The Keras software library was used to generate the illustration.

Bibliography

[1] Mark Aguiar and Gita Gopinath. Emerging market business cycles: The cycle is the trend. *Journal of Political Economy*, 115(1):69–102, 2007.

[2] K Aihara and G Matsumoto. Two stable steady states in the hodgkin-huxley axons. *Biophysical journal*, 41(1):87–89, 1983.

[3] Shun-ichi Amari. *Information geometry and its applications*, volume 194. Springer, 2016.

[4] Li An, Volker Grimm, Abigail Sullivan, BL Turner Ii, Nicolas Malleson, Alison Heppenstall, Christian Vincenot, Derek Robinson, Xinyue Ye, Jianguo Liu, et al. Challenges, tasks, and opportunities in modeling agent-based complex systems. *Ecological Modelling*, 457:109685, 2021.

[5] Sungbae An and Frank Schorfheide. Bayesian analysis of dsge models. *Econometric reviews*, 26(2-4):113–172, 2007.

[6] Joshua F Apgar, David K Witmer, Forest M White, and Bruce Tidor. Sloppy models, parameter uncertainty, and the role of experimental design. *Molecular BioSystems*, 6(10):1890–1900, 2010.

[7] W Brian Arthur. Out-of-equilibrium economics and agent-based modeling. *Handbook of computational economics*, 2:1551–1564, 2006.

[8] Amirhossein Arzani, Lingxiao Yuan, Pania Newell, and Bei Wang. Interpreting and generalizing deep learning in physics-based problems with functional linear models. *Engineering with Computers*, pages 1–23, 2024.

[9] Nooshin Asadi and Hamid Zilouei. Optimization of organosolv pretreatment of rice straw for enhanced biohydrogen production using enterobacter aerogenes. *Bioresource technology*, 227:335–344, 2017.

[10] Yuki M Asano, Jakob J Kolb, Jobst Heitzig, and J Doyne Farmer. Emergent inequality and endogenous dynamics in a simple behavioral macroeconomic model. *arXiv preprint arXiv:1907.02155*, 2019.

[11] Laurence Ball and N Gregory Mankiw. The NAIRU in theory and practice. *Journal of economic Perspectives*, 16(4):115–136, 2002.

[12] Alexander Ballantyne, Tom Cusbert, Richard Evans, Rochelle Guttmann, Jonathan Hambur, Adam Hamilton, Elizabeth Kendall, Rachael McCririck, Gabriela Nodari, and Daniel M Rees. Martin has its place: A macroeconometric model of the Australian economy. *Economic Record*, 96(314):225–251, 2020.

[13] Vera Balmer, Sophia V Kuhn, Rafael Bischof, Luis Salamanca, Walter Kaufmann, Fernando Perez-Cruz, and Michael A Kraus. Design space exploration and explanation via conditional variational autoencoders in meta-model-based conceptual design of pedestrian bridges. *Automation in Construction*, 163:105411, 2024.

[14] Aymen Balti, Valentina Lanza, and Moulay Aziz-Alaoui. A multi-base harmonic balance method applied to Hodgkin-Huxley model. *Mathematical Biosciences & Engineering*, 15(3):807–825, 2017.

[15] Dor Bank, Noam Koenigstein, and Raja Giryes. Autoencoders. *Machine learning for data science handbook: data mining and knowledge discovery handbook*, pages 353–374, 2023.

[16] Robert J Barro. Inequality, growth, and investment. Working Paper 7038, National Bureau of Economic Research, March 1999.

[17] Mohammad Mahdi Bejani and Mehdi Ghatee. A systematic review on overfitting control in shallow and deep neural networks. *Artificial Intelligence Review*, pages 1–48, 2021.

[18] Jonathan Benchimol. Money in the production function: A new Keynesian DSGE perspective. *Southern Economic Journal*, 82(1):152–184, 2015.

[19] Samy Bengio and Yoshua Bengio. Taking on the curse of dimensionality in joint distributions using neural networks. *IEEE Transactions on Neural Networks*, 11(3):550–557, 2000.

[20] Carola Conces Binder and Rodrigo Sekkel. Central bank forecasting: A survey. *Journal of Economic Surveys*, 38(2):342–364, 2024.

[21] Olivier Blanchard. Do DSGE models have a future? *DSGE Models in the Conduct of Policy: Use as Intended*, page 93, 2016.

[22] Diana Bonfim, Cláudia Custódio, and Clara Raposo. Supporting small firms through recessions and recoveries. *Journal of Financial Economics*, 147(3):658–688, 2023.

[23] George EP Box and Kenneth B Wilson. On the experimental attainment of optimum conditions. volume 13, pages 1–38. 1951.

[24] Robin Braun and Ralf Brüggemann. Identification of svar models by combining sign restrictions with external instruments. *Journal of Business & Economic Statistics*, 41(4):1077–1089, 2023.

[25] František Brázdik and Ales Marsal. Survey of research on financial sector modeling within DSGE models: what central banks can learn from it. *Available at SSRN 2274689*, 2011.

[26] William A Brock and Cars H Hommes. Heterogeneous beliefs and routes to chaos in a simple asset pricing model. *Journal of Economic dynamics and Control*, 22(8-9):1235–1274, 1998.

[27] Kevin S Brown and James P Sethna. Statistical mechanical approaches to models with many poorly known parameters. *Physical review E*, 68(2):021904, 2003.

[28] Steven L Brunton, Joshua L Proctor, and J Nathan Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15):3932–3937, 2016.

[29] Kayleigh Campbell, Laura Staugler, and Andrea Arnold. Estimating time-varying applied current in the Hodgkin-Huxley model. *Applied Sciences*, 10(2):550, 2020.

[30] Fabio Canova. Vector autoregressive models: specification, estimation, inference, and forecasting. *Handbook of applied econometrics volume 1: Macroeconomics*, pages 53–110, 1999.

[31] Fabio Canova. *Methods for applied macroeconomic research*, volume 13. Princeton university press, 2007.

[32] Yihan Cao, Siyu Li, Yixin Liu, Zhiling Yan, Yutong Dai, Philip S Yu, and Lichao Sun. A comprehensive survey of AI-generated content (AIGC): A history of generative AI from GAN to ChatGPT. *arXiv preprint arXiv:2303.04226*, 2023.

[33] Paul E Ceruzzi. *A history of modern computing*. MIT press, 2003.

[34] Marianna Cervená and Martin Schneider. Short-term forecasting gdp with a dsge model augmented by monthly indicators. Technical report, Working Paper, 2010.

[35] François Chollet et al. Keras. `https://keras.io`, 2015.

[36] Lawrence J Christiano, Martin S Eichenbaum, and Mathias Trabandt. On dsge models. *Journal of Economic Perspectives*, 32(3):113–140, 2018.

[37] Graeme S Cumming and Stephan von Cramon-Taubadel. Linking economic growth pathways and environmental sustainability by understanding development as alternate social–ecological regimes. *Proceedings of the National Academy of Sciences*, 115(38):9533–9538, 2018.

[38] Paul Inuwa Dalatu, Anwar Fitrianto, and Aida Mustapha. A comparative study of linear and nonlinear regression models for outlier detection. In *Recent Advances on Soft Computing and Data Mining: The Second International Conference on Soft Computing and Data Mining (SCDM-2016), Bandung, Indonesia, August 18-20, 2016 Proceedings Second*, pages 316–326. Springer, 2017.

[39] Sandra Daudignon and Oreste Tristani. Monetary policy and the drifting natural rate of interest. ECB Working Paper 2023/2788, European Central Bank (ECB), February 2023. Available at SSRN: `https://ssrn.com/abstract=4366502` or `http://dx.doi.org/10.2139/ssrn.4366502`.

[40] Konstantinos I Diamantaras and Sun Yuan Kung. *Principal component neural networks: theory and applications*. John Wiley & Sons, Inc., 1996.

[41] Ivo D. Dinov. *Black Box Machine-Learning Methods: Neural Networks and Support Vector Machines*, pages 383–422. Springer International Publishing, Cham, 2018.

[42] Ciobanu Dumitru and Vasilescu Maria. Advantages and disadvantages of using neural networks for predictions. *Ovidius University Annals, Series Economic Sciences*, 13(1), 2013.

[43] Romain Dupuis, Jean-Christophe Jouhaud, and Pierre Sagaut. Surrogate modeling of aerodynamic simulations for multiple operating conditions using machine learning. *Aiaa Journal*, 56(9):3622–3635, 2018.

[44] Joel Dyer, Patrick Cannon, J Doyne Farmer, and Sebastian Schmon. Black-box bayesian inference for economic agent-based models. *arXiv preprint arXiv:2202.00625*, 2022.

[45] Robert F Engle and Clive WJ Granger. Co-integration and error correction: representation, estimation, and testing. *Econometrica: journal of the Econometric Society*, pages 251–276, 1987.

[46] Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

[47] Jesús Fernández-Villaverde. The econometrics of DSGE models. *SERIEs*, 1(1-2):3–49, 2010.

[48] Jesús Fernández-Villaverde and Pablo A Guerrón-Quintana. Estimating dsge models: Recent advances and future challenges. *Annual Review of Economics*, 13(1):229–252, 2021.

[49] Jesús Fernández-Villaverde, Juan Francisco Rubio-Ramírez, and Frank Schorfheide. Solution and estimation methods for DSGE models. In *Handbook of macroeconomics*, volume 2, pages 527–724. Elsevier, 2016.

[50] Jimena Ferreira, Martín Pedemonte, and Ana I Torres. A genetic programming approach for construction of surrogate models. In *Computer Aided Chemical Engineering*, volume 47, pages 451–456. Elsevier, 2019.

[51] Alexander Forrester, Andras Sobester, and Andy Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[52] Cheng-Der Fuh, Chu-Lan Michael Kao, and Tianxiao Pang. Kullback-leibler divergence and akaike information criterion in general hidden markov models. *IEEE Transactions on Information Theory*, 2024.

[53] Hidekazu Fukai, Shinji Doi, Taishin Nomura, and Shunsuke Sato. Hopf bifurcations in multiple-parameter space of the hodgkin-huxley equations I. global organization of bistable periodic solutions. *Biological Cybernetics*, 82:215–222, 2000.

[54] Hidekazu Fukai, Taishin Nomura, Shinji Doi, and Shunsuke Sato. Hopf bifurcations in multiple-parameter space of the hodgkin-huxley equations II. singularity theoretic approach and highly degenerate bifurcations. *Biological Cybernetics*, 82:223–229, 2000.

[55] Jordi Galí. *Monetary policy, inflation, and the business cycle: an introduction to the new Keynesian framework and its applications*. Princeton University Press, 2015.

[56] Hatem Ghorbel, Nicolas Zannini, Salma Cherif, Florian Sauser, David Grunenwald, William Droz, Mahamadou Baradji, and Djamel Lakehal. Smart adaptive run parameterization (sarp): enhancement of user manual selection of running parameters in fluid dynamic simulations using bio-inspired and machine-learning techniques. *Soft Computing*, 23(22):12031–12047, 2019.

[57] Thomas Gillier and Sylvain Lenfle. Experimenting in the unknown: lessons from the manhattan project. *European Management Review*, 16(2):449–469, 2019.

[58] Goldberger and Greenspan. An efficient image similarity measure based on approximations of kl-divergence between two gaussian mixtures. In *Proceedings Ninth IEEE International conference on computer vision*, pages 487–493. IEEE, 2003.

[59] Sebastian Gomez-Gonzalez, Sergey Prokudin, Bernhard Schölkopf, and Jan Peters. Real time trajectory prediction using deep conditional generative models. *IEEE Robotics and Automation Letters*, 5(2):970–976, 2020.

[60] Pedro J Gonçalves, Jan-Matthis Lueckmann, Michael Deistler, Marcel Nonnenmacher, Kaan Öcal, Giacomo Bassetto, Chaitanya Chintaluri, William F Podlaski, Sara A Haddad, Tim P Vogels, et al. Training deep neural density estimators to identify mechanistic models of neural dynamics. biorxiv. 2019.

[61] Sven Gowal, Sylvestre-Alvise Rebuffi, Olivia Wiles, Florian Stimberg, Dan Andrei Calian, and Timothy A Mann. Improving robustness using generated data. *Advances in Neural Information Processing Systems*, 34:4218–4233, 2021.

[62] Robert B Gramacy. *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences*. CRC press, 2020.

[63] Robert B Gramacy. *Surrogates: Gaussian process modeling, design, and optimization for the applied sciences*. Chapman and Hall/CRC, 2020.

[64] Jakob Grazzini, Matteo G Richiardi, and Mike Tsionas. Bayesian estimation of agent-based models. *Journal of Economic Dynamics and Control*, 77:26–47, 2017.

[65] Stanislao Gualdi, Marco Tarzia, Francesco Zamponi, and Jean-Philippe Bouchaud. Tipping points in macroeconomic agent-based models. *Journal of Economic Dynamics and Control*, 50:29–61, 2015.

[66] John Guckenheimer and Ricardo A Oliva. Chaos in the Hodgkin–Huxley model. *SIAM Journal on Applied Dynamical Systems*, 1(1):105–114, 2002.

[67] Kristian Gundersen, Anna Oleynik, Nello Blaser, and Guttorm Alendal. Semi-conditional variational auto-encoder for flow reconstruction and uncertainty quantification from limited observations. *Physics of Fluids*, 33(1), 2021.

[68] Ryan N Gutenkunst, Joshua J Waterfall, Fergal P Casey, Kevin S Brown, Christopher R Myers, and James P Sethna. Universally sloppy parameter sensitivities in systems biology models. *PLoS computational biology*, 3(10):e189, 2007.

[69] Gary D Hansen. Indivisible labor and the business cycle. *Journal of monetary Economics*, 16(3):309–327, 1985.

[70] GM Harshvardhan, Mahendra Kumar Gourisaria, Manjusha Pandey, and Siddharth Swarup Rautaray. A comprehensive survey and analysis of generative models in machine learning. *Computer Science Review*, 38:100285, 2020.

[71] Dirk Helbing. Agent-based modeling. In *Social self-organization: Agent-based simulations and experiments to study emergent social behavior*, pages 25–70. Springer, 2012.

[72] David F Hendry. A short history of macro-econometric modelling. *Journal of Banking, Finance and Sustainable Development*, 1(1), 2020.

[73] John Hicks. IS-LM: an explanation. *Journal of Post Keynesian Economics*, 3(2):139–154, 1980.

[74] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

[75] Serhat Hosder, Layne T Watson, Bernard Grossman, William H Mason, Hongman Kim, Raphael T Haftka, and Steven E Cox. Polynomial response surface approximations for the multidisciplinary design optimization of a high speed civil transport. *Optimization and Engineering*, 2:431–452, 2001.

[76] Matteo Iacoviello. House prices, borrowing constraints, and monetary policy in the business cycle. *American economic review*, 95(3):739–764, 2005.

[77] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015.

[78] Maarten Janssen. *Microfoundations: A critical inquiry*. Routledge, 2005.

[79] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical transactions of the royal society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[80] Capers Jones. *The technical and social history of software engineering*. Addison-Wesley, 2013.

[81] Andreas Christ Sølvsten Jørgensen, Atiyo Ghosh, Marc Sturrock, and Vahid Shahrezaei. Efficient bayesian inference for stochastic agent-based models. *PLoS computational biology*, 18(10):e1009508, 2022.

[82] Joël Foramitti. Virus spread – agentpy package, 2020. [Online; accessed 30-July-2023].

[83] Chandrika Kamath. Intelligent sampling for surrogate modeling, hyperparameter optimization, and data analysis. *Machine Learning with Applications*, 9:100373, 2022.

[84] John Maynard Keynes. *A Treatise on Probability*. Macmillan and Co., London, 1921.

[85] Michael T Kiley. Output gaps. *Journal of Macroeconomics*, 37:1–18, 2013.

[86] Robert G King and Sergio T Rebelo. Resuscitating real business cycles. *Handbook of macroeconomics*, 1, 1999.

[87] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[88] Frank H Knight. Risk, uncertainty and profit. *Hart, Schaffner and Marx*, 1921.

[89] Gary Koop, M Hashem Pesaran, and Simon M Potter. Impulse response analysis in nonlinear multivariate models. *Journal of econometrics*, 74(1):119–147, 1996.

[90] Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

[91] Jakub Kudela and Radomil Matousek. Recent advances and applications of surrogate models for finite element method computations: A review. *Soft Computing*, 26(24):13709–13733, 2022.

[92] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[93] Frances Y Kuo and Ian H Sloan. Lifting the curse of dimensionality. *Notices of the AMS*, 52(11):1320–1328, 2005.

[94] Benjamin Labbé, Romain Hérault, and Clément Chatelain. Learning deep neural networks for high dimensional output problems. In *2009 International Conference on Machine Learning and Applications*, pages 63–68. IEEE, 2009.

[95] Isabel Salgado Labouriau. Degenerate hopf bifurcation and nerve impulse. *SIAM journal on mathematical analysis*, 16(6):1121–1133, 1985.

[96] Andreas Lagerqvist. Parameter estimation of a non-equilibrium asset pricing model and performance analysis of the calibration in terms of sloppiness, 2014.

[97] Thibault Laugel, Xavier Renard, Marie-Jeanne Lesot, Christophe Marsala, and Marcin Detyniecki. Defining locality for surrogates in post-hoc interpretablity. *arXiv preprint arXiv:1806.07498*, 2018.

[98] Daesoo Lee, Sara Malacarne, and Erlend Aune. Vector quantized time series generation with a bidirectional prior model. *arXiv preprint arXiv:2303.04743*, 2023.

[99] David M Levy and Nestor E Terleckyj. Effects of government r&d on private r&d investment and productivity: a macroeconomic analysis. *The Bell Journal of Economics*, pages 551–561, 1983.

[100] Liang Liang, Minliang Liu, Caitlin Martin, and Wei Sun. A machine learning approach as a surrogate of finite element analysis–based inverse method to estimate the zero-pressure geometry of human thoracic aorta. *International journal for numerical methods in biomedical engineering*, 34(8):e3103, 2018.

[101] Edward N Lorenz. Deterministic nonperiodic flow. *Journal of the atmospheric sciences*, 20(2):130–141, 1963.

[102] Robert E Lucas Jr. Econometric policy evaluation: A critique. In *Carnegie-Rochester conference series on public policy*, volume 1, pages 19–46. North-Holland, 1976.

[103] K Lukošiute, Geert Raaijmakers, Zoheyr Doctor, Marcelle Soares-Santos, and Brian Nord. Kilonovanet: Surrogate models of kilonova spectra with conditional variational autoencoders. *Monthly Notices of the Royal Astronomical Society*, 516(1):1137–1148, 2022.

[104] Charles M Macal and Michael J North. Tutorial on agent-based modeling and simulation. In *Proceedings of the Winter Simulation Conference, 2005.*, pages 14–pp. IEEE, 2005.

[105] G Maione. Continued fractions approximation of the impulse response of fractional-order dynamic systems. *IET Control Theory & Applications*, 2(7):564–572, 2008.

[106] Brian K Mannakee, Aaron P Ragsdale, Mark K Transtrum, and Ryan N Gutenkunst. Sloppiness and the geometry of parameter space. *Uncertainty in Biology: A Computational Modeling Approach*, pages 271–299, 2016.

[107] Michael D McKay, Richard J Beckman, and William J Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 42(1):55–61, 2000.

[108] Umberto Michelucci. An introduction to autoencoders. *arXiv preprint arXiv:2201.03898*, 2022.

[109] Sebastian Mika, Bernhard Schölkopf, Alex Smola, Klaus-Robert Müller, Matthias Scholz, and Gunnar Rätsch. Kernel pca and de-noising in feature spaces. *Advances in neural information processing systems*, 11, 1998.

[110] Gloria M Monsalve-Bravo, Brodie AJ Lawson, Christopher Drovandi, Kevin Burrage, Kevin S Brown, Christopher M Baker, Sarah A Vollert, Kerrie Mengersen, Eve McDonald-Madden, and Matthew P Adams. Analysis of sloppiness in model simulations: Unveiling parameter uncertainty when mathematical models are fitted to data. *Science advances*, 8(38):eabm5952, 2022.

[111] John T Nardini, Ruth E Baker, Matthew J Simpson, and Kevin B Flores. Learning differential equation models from stochastic agent-based model simulations. *Journal of the Royal Society Interface*, 18(176):20200987, 2021.

[112] Karl Naumann-Woleske, Max Sina Knicker, Michael Benzaquen, and Jean-Philippe Bouchaud. Exploration of the parameter space in macroeconomic agent-based models. *arXiv preprint arXiv:2111.08654*, 2021.

[113] Mario Negre, José Cuesta, Ana Revenga, and Prescott J Morley. Dismantling the myth of the growth-inequality trade-off. Technical report, Briefing Paper, 2019.

[114] Frank Nielsen. The many faces of information geometry. *Not. Am. Math. Soc*, 69(1):36–45, 2022.

[115] Chandramouli Nyshadham, Matthias Rupp, Brayden Bekker, Alexander V Shapeev, Tim Mueller, Conrad W Rosenbrock, Gábor Csányi, David W Wingate, and Gus LW Hart. Machine-learned multi-system surrogate models for materials prediction. *npj Computational Materials*, 5(1):51, 2019.

[116] Johannes Pfeifer. DSGE_mod: A collection of Dynare models (release 1.0.1), 2021. Available at: `https://github.com/JohannesPfeifer/DSGE_mod`.

[117] Donovan Platt. A comparison of economic agent-based model calibration methods. *Journal of Economic Dynamics and Control*, 113:103859, 2020.

[118] Sebastian Poledna, Michael Gregor Miess, Cars Hommes, and Katrin Rabitsch. Economic forecasting with an agent-based model. *European Economic Review*, 151:104306, 2023.

[119] Leah F Price, Christopher C Drovandi, Anthony Lee, and David J Nott. Bayesian synthetic likelihood. *Journal of Computational and Graphical Statistics*, 27(1):1–11, 2018.

[120] Ville Pulkki and Juha Merimaa. Spatial impulse response rendering II: Reproduction of diffuse sound and listening tests. *Journal of the Audio Engineering Society*, 54(1/2):3–20, 2006.

[121] Katherine N Quinn, Michael C Abbott, Mark K Transtrum, Benjamin B Machta, and James P Sethna. Information geometry for multiparameter models: New perspectives on the origin of simplicity. *Reports on Progress in Physics*, 2022.

[122] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[123] David Romer. *Advanced macroeconomics*. A Division of The McGraw-Hill Companies, 4th edition, 2011.

[124] Paul A. Samuelson and William D. Nordhaus. *Economics*. McGraw-Hill, 17th edition, 2001.

[125] Thomas J Sargent and Lars Ljungqvist. Recursive macroeconomic theory. *Massachusetss Institute of Technology*, 2000.

[126] Stephanie Schmitt-Grohé and Martın Uribe. Closing small open economy models. *Journal of international Economics*, 61(1):163–185, 2003.

[127] Frank Schorfheide. Estimation and evaluation of DSGE models: progress and challenges. 2011.

[128] Matthias Seeger. Gaussian processes for machine learning. *International journal of neural systems*, 14(02):69–106, 2004.

[129] Francesco Sergi. The standard narrative about DSGE models in central banks' technical reports. *The European journal of the history of economic thought*, 27(2):163–193, 2020.

[130] Dhruv Sharma, Jean-Philippe Bouchaud, Stanislao Gualdi, Marco Tarzia, and Francesco Zamponi. V–, U–, L–or W–shaped economic recovery after Covid-19: Insights from an agent based model. *PloS one*, 16(3):e0247823, 2021.

[131] Kamran Shaukat, Farhat Iqbal, Talha Mahboob Alam, Gagandeep Kaur Aujla, Liton Devnath, Abdul Ghaffar Khan, Rimsha Iqbal, Irum Shahzadi, and Afifah Rubab. The impact of artificial intelligence and robotics on the future employment opportunities. *Trends in Computer Science and Information Technology*, 5(1):050–054, 2020.

[132] Michael D Shields and Jiaxin Zhang. The generalization of latin hypercube sampling. *Reliability Engineering & System Safety*, 148:96–108, 2016.

[133] Christopher A Sims. Are forecasting models usable for policy analysis? *Quarterly Review*, 10(Win):2–16, 1986.

[134] Frank Smets, Kai Christoffel, Günter Coenen, Roberto Motto, and Massimo Rostagno. DSGE models and their use at the ECB. *SERIEs*, 1:51–65, 2010.

[135] Frank Smets and Raf Wouters. An estimated dynamic stochastic general equilibrium model of the euro area. *Journal of the European economic association*, 1(5):1123–1175, 2003.

[136] Frank Smets and Rafael Wouters. Shocks and frictions in us business cycles: A Bayesian DSGE approach. *American economic review*, 97(3):586–606, 2007.

[137] Adam Smith. *The wealth of nations [1776]*, volume 11937. na, 1937.

[138] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28, 2015.

[139] Mario Solis-Garcia. A (pedagogical) note on the log-linearization of functions of several variables. *International Review of Economics Education*, 37:100210, 2021.

[140] James C Spall. Monte carlo computation of the fisher information matrix in nonstandard settings. *Journal of Computational and Graphical Statistics*, 14(4):889–909, 2005.

[141] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[142] David M Steinberg and Dennis KJ Lin. A construction method for orthogonal latin hypercube designs. *Biometrika*, 93(2):279–288, 2006.

[143] Petre Stoica and Yngve Selen. Model-order selection: a review of information criterion rules. *IEEE Signal Processing Magazine*, 21(4):36–47, 2004.

[144] Amir Sufi and Alan M. Taylor. Chapter 7 - financial crises: a survey. In Gita Gopinath, Elhanan Helpman, and Kenneth Rogoff, editors, *Handbook of International Economics: International Macroeconomics, Volume 6*, volume 6 of *Handbook of International Economics*, pages 291–340. Elsevier, 2022.

[145] Vanja G Švenda, Mark K Transtrum, Benjamin L Francis, Andrija T Sarić, and Aleksandar M Stanković. State estimation model reduction through the manifold boundary approximation method. *IEEE Transactions on Power Systems*, 37(1):272–281, 2021.

[146] Vajira Thambawita, Pegah Salehi, Sajad Amouei Sheshkal, Steven A Hicks, Hugo L Hammer, Sravanthi Parasa, Thomas de Lange, Pål Halvorsen, and Michael A Riegler. SinGAN-Seg: Synthetic training data generation for medical image segmentation. *PloS one*, 17(5):e0267976, 2022.

[147] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society Series B: Statistical Methodology*, 58(1):267–288, 1996.

[148] Mark K Transtrum, Benjamin B Machta, Kevin S Brown, Bryan C Daniels, Christopher R Myers, and James P Sethna. Perspective: Sloppiness and emergent theories in physics, biology, and beyond. *The Journal of chemical physics*, 143(1):07B201_1, 2015.

[149] Mark K Transtrum, Benjamin B Machta, and James P Sethna. Why are nonlinear fits to data so challenging? *Physical review letters*, 104(6):060201, 2010.

[150] Mark K Transtrum, Benjamin B Machta, and James P Sethna. Geometry of nonlinear least squares with applications to sloppy models and optimization. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics*, 83(3):036701, 2011.

[151] Mark K Transtrum and Peng Qiu. Model reduction by manifold boundaries. *Physical review letters*, 113(9):098701, 2014.

[152] Alessandro Vercelli. Microfoundations, methodological individualism and alternative economic visions. *Review of Political Economy*, 28(1):153–167, 2016.

[153] David Vines and Samuel Wills. The rebuilding macroeconomic theory project part ii: multiple equilibria, toy models, and policy models in a new macroeconomic paradigm. *Oxford Review of Economic Policy*, 36(3):427–497, 2020.

[154] Tian Wang, Ye Wang, Jiamin Shen, Lei Wang, and Lihong Cao. Predicting spike features of hodgkin-huxley-type neurons with simple artificial neural network. *Frontiers in computational neuroscience*, 15:800875, 2022.

[155] Joshua J Waterfall, Fergal P Casey, Ryan N Gutenkunst, Kevin S Brown, Christopher R Myers, Piet W Brouwer, Veit Elser, and James P Sethna. Sloppy-model universality class and the vandermonde matrix. *Physical review letters*, 97(15):150601, 2006.

[156] Dominic G Whittaker, Jiahui Wang, Joseph G Shuttleworth, Ravichandra Venkateshappa, Jacob M Kemp, Thomas W Claydon, and Gary R Mirams. Ion channel model reduction using manifold boundaries. *Journal of the Royal Society Interface*, 19(193):20220193, 2022.

[157] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.

[158] Piotr Iwo Wójcik and Marcin Kurdziel. Training neural networks on high-dimensional data using random projection. *Pattern Analysis and Applications*, 22:1221–1231, 2019.

[159] Takeshi Yagihashi. Dsge models used by policymakers: A survey. *Policy Research Institute, Ministry of Finance, Japan, PRI Discussion Paper Series*, 2020.

[160] Kazuo Yonekura and Katsuyuki Suzuki. Data-driven design exploration method using conditional variational autoencoder for airfoil design. *Structural and Multidisciplinary Optimization*, 64(2):613–624, 2021.

[161] J Zhang and FTK Au. Calibration of initial cable forces in cable-stayed bridge based on kriging approach. *Finite Elements in Analysis and Design*, 92:80–92, 2014.