Durham University

Durham E-Theses

Policy making using computer simulators for complex physical systems; Bayesian decision support for the development of adaptive strategies

WILLIAMSON, DANIEL

How to cite:

WILLIAMSON, DANIEL (2010) *Policy making using computer simulators for complex physical systems; Bayesian decision support for the development of adaptive strategies*, Durham theses, Durham University. Available at Durham E-Theses Online: http://etheses.dur.ac.uk/348/

Use policy

The full-text may be used and/or reproduced, and given to third parties in any format or medium, without prior permission or charge, for personal research or study, educational, or not-for-profit purposes provided that:

- a full bibliographic reference is made to the original source
- a link is made to the metadata record in Durham E-Theses
- the full-text is not changed in any way

The full-text must not be sold in any format or medium without the formal permission of the copyright holders.

Please consult the full Durham E-Theses policy for further details.

Academic Support Office, The Palatine Centre, Durham University, Stockton Road, Durham, DH1 3LE
e-mail: e-theses.admin@durham.ac.uk Tel: +44 0191 334 6107
http://etheses.dur.ac.uk

Policy making using computer simulators for complex physical systems; Bayesian decision support for the development of adaptive strategies

Daniel Williamson

A Thesis presented for the degree of
Doctor of Philosophy

Statistics and Probability Group
Department of Mathematical Sciences
University of Durham
England

March 2010

Policy making using computer simulators for complex physical systems; Bayesian decision support for the development of adaptive strategies

Daniel Williamson

Submitted for the degree of Doctor of Philosophy
March 2010

Abstract

Policy makers increasingly rely on computer models to aid policy judgements for complex systems. The climate system, for example, is extremely complicated and its reaction to changes in radiative forcing through CO_2 emissions can only be explored using models. Bayesian methods for making inferences about physical systems that combine information from computer simulators and system observations have become increasingly well studied. We apply some of these methods to the policy problem where the decisions to be made are inputs to the computer model. Particular features of our methodologies include: the provision of Bayesian decision support for the policy problem when it is known that policy may be adapted in reaction to future observations of the complex system; and careful integration of the knowledge that our computer simulators will evolve and improve over time, which may affect downstream strategies and, hence, current policy.

Our methods also allow research investment questions to be explored in the context of the wider policy problem. For example, the question of whether or not an improved version of a computer simulator should be built and how much it should be run can be addressed as part of the policy problem.

Declaration

The work in this thesis is based on research carried out at the Statistics and Probability Group, the Department of Mathematical Sciences, Durham University, England. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Copyright © 2010 by Daniel Williamson.

"The copyright of this thesis rests with the author. No quotations from it should be published without the author's prior written consent and information derived from it should be acknowledged".

Acknowledgements

There are a number of people I must thank for helping me produce this work. First and foremost I would like to thank my supervisor Michael Goldstein. Without his help and guidance this work would not have been possible. I would also like to thank the people at EPSRC who funded me. Many thanks go to Peter Challenor at the University of Southampton for suggesting that we consider the CO_2 abatement problem and for the guidance he provided when constructing the example. I must also thank Gemma Stephenson, Peter's phd student, for spending a great deal of time helping me to design a batch of runs for C-GOLDSTEIN and then performing the runs for me.

To my wife Becky, who not only provided day to day support, but proof read the whole thesis to check my spelling and grammar, I will always be grateful. I must also thank Peter Craig and Jonathan Cumming for a number of useful discussions over coffee or a white board, and Becky O'Neil for the countless chats and favours during three years of sharing an office. There are so many other people I could mention, but then these acknowledgements might become as long as the thesis. Suffice it to say that if you were nice to me at all during these three and a half years, I am grateful. I'd finally like to thank both of my parents for helping me get this far and for the unconditional support they have given me throughout my 21 years of education.

Contents

Abstract **ii**

Declaration **iii**

Acknowledgements **iv**

1 Introduction **1**

1.1 Thesis outline 2

2 Learning about complex systems using computer models **4**

2.1 On the use of computer models describing physical systems 5

2.1.1 Inputs and outputs of the computer code 6

2.1.2 Relating models and systems 7

2.2 Emulators 10

2.2.1 Justifying a stochastic approach 10

2.2.2 A general form of emulator 11

2.2.3 Kriging 12

2.2.4 Bayesian emulation 15

2.2.5 Bayes Linear Emulation 16

2.2.6 Practical emulation for large-scale models 22

2.3 Using emulators to study physical systems 25

2.3.1 History Matching 25

2.3.2 Forecasting 27

2.3.3 Separability 33

2.3.4 Fast Forecasting 36

2.4 Design and other topics 40
2.4.1 Design of computer experiments 40
2.4.2 Emulator Diagnostics 43

3 Model aided decision support for policy problems with intervention 44
3.1 The policy problem with intervention 45
3.1.1 Loss models 45
3.1.2 Optimal policy and intervention 46
3.1.3 Formalizing the policy problem 46
3.2 Integrated Assessment 47
3.2.1 Locating optimal policy 49
3.2.2 Investigating adaptive strategies 50
3.2.3 Handling and reporting uncertainty 50
3.2.4 Drawbacks to the IA approach 51
3.3 The expected loss surface 53
3.3.1 The ideal solution 53
3.3.2 Obtaining estimates to the required distributions 57
3.3.3 Evaluating expectations 59
3.3.4 Combining forecasts 60
3.4 Multi-level emulation 61
3.4.1 Multi-level emulation of computer models 61
3.4.2 Emulation of expected losses 64
3.4.3 Bayes Linear calculations for multi-level emulation 65
3.5 Sequential Emulation 68
3.5.1 Strategy at t_m 68
3.5.2 Removing the next decision node 70
3.5.3 Determining time t_k intervention strategy 71
3.5.4 The upper bound on our expected loss surface 72
3.6 The Sequential Emulation Algorithm 73
3.6.1 Preliminaries 73
3.6.2 Sequential Emulation 75
3.7 Policy Support 77

3.7.1 Pruning 77
3.7.2 Sensitivity and scenario analysis 82
3.7.3 Risk profiling 84
3.8 Discussion 85
3.8.1 Feasibility of Sequential Emulation 85
3.8.2 Treatment of the loss model 89

4 Sequential Emulation for a simplified climate policy problem 91
4.1 The CO_2 emissions problem 92
4.2 The Computer Models 94
4.2.1 C-GOLDSTEIN 94
4.2.2 DICE 95
4.3 Emulation of C-GOLDSTEIN 97
4.3.1 Model runs and the form of the emulator 98
4.3.2 Choosing the regressors 99
4.3.3 Constructing a second-order prior for β 103
4.3.4 The residual process $u(x, \theta)$ 106
4.3.5 Completing our emulation 113
4.4 DICE as a loss function 116
4.5 Completing the preliminaries 120
4.5.1 Obtaining forecasts 120
4.5.2 Characterizing the required distributions 121
4.5.3 Coarse and accurate evaluations 122
4.6 Sequential Emulation 123
4.6.1 Designs 123
4.6.2 Building the coarse models 124
4.6.3 Emulating $A(\theta^1, z^1)$ 126
4.6.4 Defining time t_1 strategy, λ_{t_1} 131
4.6.5 Emulating $B^1_{\lambda^1}(\theta_{t_0}, z^1)$ 132
4.6.6 Emulating $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$ 137
4.7 Policy support 140
4.7.1 Forward sampling 140

4.7.2 Pruning 142
4.7.3 Risk profiling 150
4.7.4 Sensitivity analysis 152
4.7.5 Relating expected loss to cost 159

5 Policy support with evolving computer models 161

5.1 Evolving computer models 162
5.2 Reification 163
5.2.1 The best input re-visited 163
5.2.2 The Reified Simulator 165
5.3 The policy problem with evolving simulators 168
5.4 Relating simulators 169
5.4.1 Relating emulators 172
5.5 Model-related observations 173
5.5.1 Future model runs 173
5.5.2 An alternative treatment of future model observations 174
5.6 Reified decision-dependent forecasts 177
5.6.1 Adjusting the reified coefficients 177
5.6.2 An alternative model 181
5.6.3 Reified forecasting calculations 184
5.6.4 Decision-dependent observation forecasts 185
5.7 Sequential Emulation with evolving models 188
5.7.1 Reified Sequential Emulation algorithm 189
5.8 Illustrative example 192
5.8.1 Structural reification of C-GOLDSTEIN 193
5.8.2 Decisions regarding the improved model 196
5.8.3 Completing the preliminaries 198
5.8.4 Calculations 199
5.8.5 Additional policy support 201
5.9 Discussion 203
5.9.1 Feasibility of Reified Sequential Emulation 203
5.9.2 Further policy support 205

6 Bayes Linear calibrated decision-dependent forecasts 207

6.1 The Hat run 208

6.1.1 The hat run moments via sampling 211

6.2 The Hat function 214

6.2.1 The Hat spin 215

6.2.2 Forecasting via the hat function 216

6.2.3 Sampling the hat function moments 219

6.2.4 Using a computer algebra package 221

6.2.5 Dealing with $u(\hat{x}, \theta)$ 223

6.2.6 The practicalities of using a computer algebra package 226

6.3 Sequential Emulation with the hat function 229

6.3.1 Just one hat function 230

6.3.2 Future hat functions 232

7 Concluding remarks 238

Bibliography 243

Appendix 254

A Notation 254

A.1 Use of subscripts and the manipulation of arrays 254

A.2 Nomenclature 255

B Integrating out the best input of C-GOLDSTEIN analytically 262

C The DICE model 267

D Selected annotated code for forecasting and Sequential Emulation 273

D.1 Decision-dependent forecasting 273

D.1.1 The prior emulator 274

D.1.2 Adjusting the prior emulator 277

D.1.3 Integrating out x^* 281

D.1.4 Obtaining the beliefs about y 283

D.1.5 Decision-dependent forecast 289
D.1.6 Wrappers 291
D.2 Sequential Emulation 293
D.2.1 Sampling from log-normal distributions 293
D.2.2 Multi-level adjustment 295
D.2.3 Obtaining coarse and accurate runs 300
D.2.4 The Final Emulators 304
D.3 Reified decision-dependent forecasting 308

E Miscellaneous details of built emulators 317
E.1 The Emulator for C-GOLDSTEIN 317
E.2 Sequential Emulation in Chapter 4 319
E.2.1 The emulator of section 4.6.3 319
E.2.2 The emulator of section 4.6.5 324
E.2.3 The emulator of section 4.6.6 327
E.3 Policy Support in Chapter 4 329
E.3.1 More strategy plots 329
E.3.2 Details of the Sequential Emulation on the pruned space . . . 329
E.4 The Reified Sequential Emulation example 336

F Computing the hat function moments with a computer algebra package 347

List of Figures

3.1 Our statement of the decision problem and the modelling statements made in chapter 2 define an influence diagram. This figure presents an example of this influence diagram for the case where we have 2 downstream intervention points at times t_1 and t_2. 48

3.2 A snapshot of the decision tree defined by the intervention problem when $m = 1$. The node labels correspond to the quantities defined by (3.1), (3.2) and (3.3). The dotted lines represent the infinite number of alternative paths defined by the different decisions and observations we may make. 54

3.3 This image shows the decision tree for the full intervention problem after the expected losses as defined by (3.1) have been computed for each branch defined by θ^m and z^m. The dashed line between the two solid chance nodes represents the $m - 1$ decisions and observations that are taken before we observe z_{t_m}. 55

3.4 This image shows the decision tree from figure 3.3 having been rolled back one step by choosing the θ_{t_m} that minimizes $A(\theta^m, z^m)$ for each θ^{m-1}, z^m. 56

3.5 This image shows the decision tree from figure 3.4 rolled back a further step, by taking expectations of $B^m(\theta^{m-1}, z^m)$ over the conditional distribution of z_{t_m} given all other decisions and observations, as defined by (3.3). 56

3.6 This image shows our decision tree after we have fixed $\lambda_{t_m}(\theta^{m-1}, z^m)$. The right-most node is now replaced by the upper bound $B^m_{\lambda^m}(\theta^{m-1}, z^m)$ as defined in (3.17). 70

3.7 This image shows a sub tree of our decision tree, starting from the kth decision, after we have fixed our first $m-k$ intervention strategies, thus removing the corresponding decision nodes. The dashed line connecting the two right-most chance nodes represents observations $z_{t_{k+2}}, \ldots, z_{t_m}$ 72

3.8 A flow chart describing the steps of the Sequential Emulation algorithm. The labels P1-P5 refer to the preliminary steps of the algorithm. The labels S1-S10 refer to the main steps of the algorithm. . . 78

4.1 Pairs graph for the model output 1995 temperature anomaly, F_1, for different values of the model input variables x_1, x_2, and x_3. 100

4.2 Pairs graph for the model output 2035 temperature anomaly F_2 for different values of the model input variables x_1, x_2, x_3 and the decision variable θ_{t_0}. 101

4.3 Pairs graph for the model output 2095 temperature anomaly F_3 for different values of the model input variables x_1, x_2, x_3 and the decision variables θ_{t_0} and θ_{t_1}. 102

4.4 Leave one out diagnostic graph used to fix σ_1. The dotted red lines represent ± 2 standard deviations with respect to the original linear model. Each of the points represents the difference between the data point left out and the prediction for it using a linear model fitted to the rest of the data. 108

4.5 Leave one out diagnostic graph used to fix σ_2. The dotted red lines represent ± 2 standard deviations with respect to the original linear model. 109

4.6 Leave one spin up out diagnostic graph used to fix σ_2. The dotted red lines represent ± 2 standard deviations with respect to the original linear model. 109

4.7 Leave one out diagnostic graph used to fix σ_3. The dotted red lines represent ± 2 standard deviations with respect to the original linear model. 110

4.8 Leave one spin up out diagnostic graph used to fix σ_3. The dotted red lines represent ± 2 standard deviations with respect to the original linear model. 111

4.9 This picture shows two potential fits (represented by the lines) to a set of points. We argue in the text that with certain fitted variances and correlation parameters we could build two emulators that both go through the points in exactly the same way. 112

4.10 Typical quadratic curve 112

4.11 Leave one out diagnostic graph used to check the performance of our adjusted emulator for $f_1(x)$. 114

4.12 Leave one out diagnostic graph used to check the performance of our adjusted emulator for $f_2(x, \theta)$. 115

4.13 Leave one out diagnostic graph used to check the performance of our adjusted emulator for $f_3(x, \theta)$. 116

4.14 A plot of the model data for 2035 against θ_{t_0}. We colour each spin up black, red, and blue alternately. This plot demonstrates that the relationship between the model output and θ_{t_0} is roughly the same for each spin up. 117

4.15 Scatter plots of DICE output for a Latin Hypercube Design on the policies and temperatures. Loss here is evaluated as negative utility. . 118

4.16 Two scatter plots of DICE output for different policies with the temperature input fixed. Loss here is evaluated as negative utility. 119

4.17 Scatter plots of our coarse evaluations of $A(\theta^1, z^1)$. 127

4.18 Residual plots from fitting a saturated linear model of order two to the coarse data for $A(\theta^1, z^1)$. 128

4.19 Residual plots from fitting a saturated linear model of order three to the coarse data for $A(\theta^1, z^1)$. 130

4.20 A Strategy plot for θ_{t_1}, where the surface $\lambda_{t_1}(\theta_{t_0}, \theta^1)$ is plotted as a function of θ_{t_0} and z_{t_1}. 132

4.21 Scatter plots of our coarse evaluations of $B^1_{\lambda^1}(\theta_{t_0}, z^1)$. 133

4.22 Residual plots from fitting a saturated linear model of order two to the coarse data for $B^1_{\lambda^1}(\theta_{t_0}, z^1)$. 134
4.23 Residual plots from fitting a saturated linear model of order three to the coarse data for $B^1_{\lambda^1}(\theta_{t_0}, z^1)$. 136
4.24 A scatter plot of our coarse evaluations of $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$. 137
4.25 Residual plots from fitting $\theta_{t_0} + \theta^2_{t_0}$ to the coarse data for $C^1_{\lambda^1}(\theta_{t_0}, z^1)$. 138
4.26 Residual plots from fitting $\theta_{t_0}+\theta^2_{t_0}+\theta^3_{t_0}$ to the coarse data for $C^1_{\lambda^1}(\theta_{t_0}, z^1)$.139
4.27 A plot of $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ with error bars obtained through the forward sampling methods described in section 4.7.1. 141
4.28 Strategy plot for $\theta_{t_0} = arg\min_{\theta_{t_0}}\{\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]\}$. 144
4.29 Panel of strategy plots for $\theta_{t_0} = 0.2$. Each panel corresponds to one sampled z_{t_1}. Red dotted lines on each panel indicate $\theta_{t_1} = -0.65$ and $\theta_{t_1} = 0.65$. 145
4.30 The image of $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ with the blue points indicating where our expected loss would be if we chose an intervention strategy that obtained the highest maximum observed potential improvement observed from all of our samples. 146
4.31 The image of $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ with the blue points indicating where our expected loss would be if we chose an intervention strategy that obtained the highest maximum observed potential improvement observed from all of our samples. Yellow dotted lines indicate the area we intend to focus our sampling, whilst the other lines are aids for pruning. 147
4.32 A plot of $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ for the Sequential Emulation performed on the pruned decision tree. 149
4.33 Four histograms showing the risk profile for $\theta_{t_0} = arg\min_{\theta_{t_0}}\{\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]\}$ under our original Sequential Emulation (before pruning). Each panel shows an increasingly narrow subset of the possible losses in the risk profile. 151

4.34 Four histograms showing the risk profile for $\theta_{t_0} = 1$ under our original Sequential Emulation (before pruning). Each panel shows an increasingly narrow subset of the possible losses in the risk profile. . . 153

4.35 Results of Sequential Emulation under three different parametrizations of DICE. The central plot represents our original Sequential Emulations. The first plot has $r = 0.005$ and the final plot has $g^b(0) = 0.35$. 154

4.36 Results of Sequential Emulation with three different discrepancy variances for our original parametrization of DICE. 156

4.37 Results of Sequential Emulation with three different discrepancy variances for our parametrization of DICE with higher utility for the wealth of future generations. 157

4.38 Results of Sequential Emulation with three different discrepancy variances for our parametrization of DICE with large immediate abatement costs. 158

4.39 A plot of the annual differences in global consumption for two forward samples taken from $\theta_{t_0} \approx 0.7$. The first has expected loss of 0.145 and the second has expected loss 0.155. 160

5.1 Our statement of the decision problem and the modelling statements we have made define an influence diagram. This figure presents an example of this influence diagram for the case where we have 2 downstream intervention points at times t_1 and t_2, and the option to build and run an improved version of our current model at each of these times. To make the research investment decisions clear on the diagram, we give them separate nodes and label them R_1 and R_2. These represent the decisions of whether or not to build and how much to run each new model. 170

5.2 This image shows the decision tree for the case where we may build and run new models at each of the m intervention points. Observations of new models are characterized by observed adjusted coefficients, H^k, for the regression surfaces of our m model emulators, for $k = 1, \ldots, m$. The function $A(\theta^m, z^m, H^{[m]})$, defined formally below by equation (5.32), is our expected loss for having taken decisions θ^m and having observed $\{z^m, H^{[m]}\}$. 176

5.3 A plot of our emulated upper bound $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ where $Var\left[\rho'\right] = 0.25$ and $Var\left[\rho^*\right] = 0.1$. 201

5.4 A plot of our emulated upper bound $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ where $Var\left[\rho'\right] = 0.1$ and $Var\left[\rho^*\right] = 0.25$. 202

6.1 An example of the decision tree defined by the policy problem with one intervention point that includes the option of performing the hat spin and evaluating the hat function defined by any z_{t_1}. Although the tree marks the observation of $\hat{f}^1$ if we choose to perform the hat spin, we argue in the main text that this observation is a surface in θ_{t_1}. This surface is not observed in its entirety having performed the hat spin, but we are able to evaluate it for many θ_{t_1} by evaluating the hat function. 233

E.1 Panel of strategy plots for $\theta_{t_0} = arg\min_{\theta_{t_0}}\{\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]\}$. Each panel corresponds to one sampled z_{t_1}. Red dotted lines on each panel indicate $\theta_{t_1} = -0.65$ and $\theta_{t_1} = 0.65$. 330

E.2 Panel of strategy plots for $\theta_{t_0} = 0.95$. Each panel corresponds to one sampled z_{t_1}. Red dotted lines on each panel indicate $\theta_{t_1} = -0.65$ and $\theta_{t_1} = 0.65$. 331

E.3 Panel of strategy plots for $\theta_{t_0} = 0.4$. Each panel corresponds to one sampled z_{t_1}. Red dotted lines on each panel indicate $\theta_{t_1} = -0.65$ and $\theta_{t_1} = 0.65$. 332

E.4 Time t_1 strategy plot for θ_{t_1}, where the surface $\lambda_{t_1}(\theta_{t_0}, \theta^1)$, as calculated via our emulator for $A(\theta^1, z^1)$ on the pruned decision space, is plotted as a function of θ_{t_0} and z_{t_1}. 334

E.5 The coarse data used for building an emulator for $A(\theta^1, z^1, H^1)$. . . . 337

E.6 Residual plots from fitting a saturated linear model of order three to the coarse data for $A(\theta^1, z^1, H^1)$. 341

E.7 The coarse data used for building an emulator for $B^1_{\lambda^1}(\theta_{t_0}, z^1, H^1)$. . . 342

E.8 The coarse data used for building an emulator for $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0}, H^1)$. . . 346

List of Tables

4.1 Prior expectations for the regression coefficients in our emulator for C-Goldstein, rounded to 2 significant figures. 104
4.2 Prior standard deviations for the regression coefficients in our emulator for C-Goldstein, rounded to 2 significant figures. 105

E.1 Adjusted expectations for the regression coefficients in our emulator for C-Goldstein. 317
E.2 Adjustment discrepancy for the regression coefficients in our emulator for C-Goldstein. 318

Chapter 1

Introduction

Computer models are now used in many areas of science to investigate the behaviour of complex physical systems. There are many different applications for the study of complex systems using computer models, and in this thesis we focus on one in particular. We consider the case where policy decisions must be made in order to influence future states of complex physical systems and look at how computer models for those systems may be used in order to learn about the possible behaviour of the system under a given policy. There are many examples of such problems arising from the issues surrounding climate change. Many difficult policy decisions regarding CO_2 abatement and investment into alternative energies must be taken in response to climate change, and the principle method for studying climate is through the use of large-scale computer models.

Managing uncertainty in computer models and using them to make inference regarding complex systems is a relatively new field of study. In this thesis we shall look at applying some of the recent and sophisticated Bayesian methods in this area in order to provide decision support for complex policy problems. Our methods move away from Integrated Assessment and carefully handle the discrepancy between the model and the system. We allow future observations of the system to be made and downstream strategy to be adapted accordingly. We also provide a treatment for the policy problem where our computer models are allowed to evolve and improve over time.

As part of our study of this problem, we aim to generalize some of the current

Bayes Linear technologies for forecasting future states of complex systems using computer models, to the case where the model output and system state depend on decision variables.

1.1 Thesis outline

In chapter 2 we provide a review of some of the current statistical technologies for making inference about complex physical systems using computer models. We also generalize the Bayes Linear forecasting methods of Craig et al. [17] in order to define Bayes Linear decision-dependent forecasts. We discuss the computational issues that arise in the case where each forecast depends on decision variables and present a number of techniques designed to tackle these issues.

In chapter 3 we define the policy problem in which we are interested and provide a review of some of the current decision support methods available for these problems. We then show how current methods in forecasting future states of complex systems using computer models can be used in order to derive the probabilities on an infinite decision tree. We develop a technique that we call Sequential Emulation, designed to mimic the backwards induction process and give insight into our expected loss surface for this problem. We present a number of ideas for using the results of a Sequential Emulation to provide decision support for policy makers in section 3.7, and discuss the feasibility of the methodology in section 3.8. In chapter 4 we illustrate the implementation of our methods using a simplified version of a real-world policy problem related to climate change.

In chapter 5 we extend the policy problem that we consider to the case where our computer models improve and where we may have information from, as yet unbuilt, future models when we come to make policy interventions. We apply the structural reification ideas of Goldstein and Rougier [40] in order to provide a belief framework for considering future models in the context of our current model. We introduce a pragmatic approach to quantifying observable information on future models and use this approach to define forecasts that are appropriately influenced by this observed information where it is available. We generalize our Sequential Emulation approach

in order to give insight into the expected loss surface for this decision tree, and describe the ways in which our methods may also be applied to research investment decisions. In particular, the question of whether or not a specific improved version of a simulator should be built and how much that simulator should be run is given formal treatment. In section 5.8 we illustrate some of these methods by extending the example we introduced in chapter 4.

In chapter 6 we return to the case where we have only one model to consider, and look to generalize the Bayes Linear calibrated forecasting methodology of Goldstein and Rougier [39] to the case where we have decisions that affect future states of a complex system as inputs to our model. We describe how historical observations of the physical system define a hat function and argue that, for models with a spin up property, the hat function could be a useful tool for resolving local model uncertainty around the best input. We describe Bayes Linear calibrated decision-dependent forecasts and offer an in-depth discussion regarding the practicalities of computing them. Finally, we discuss the use of Bayes Linear calibrated decision-dependent forecasts as part of a Sequential Emulation for the policy problem.

Chapter 7 offers concluding remarks and highlights a number of potential areas for further work. Appendices to this thesis present some of the computer code used in implementing our methods in R. Appendix A describes the notational conventions used in this thesis and also contains a detailed glossary of the notation.

Chapter 2

Learning about complex systems using computer models

In this chapter we give a detailed review of the theory and practice involved in using computer models to learn about complex physical systems. In section 2.1 we present a number of ways in which computer models are used in modern science, and describe the basic properties of computer simulators and how they may be linked to reality. In section 2.2 we introduce the concept of an emulator for a computer model and review the main concepts and practices involved when emulating computer code. Section 2.3 presents some of the ways in which emulators may be used to study complex physical systems. The principle focus of this section will be using an emulator to forecast future behaviour of the system. In sections 2.3.3 and 2.3.4 we present a number of ideas and techniques designed to simplify and expidite Bayes Linear forecasting calculations for the case where decisions to be taken that will influence the behaviour of the system are also inputs to our model. We conclude the chapter in section 2.4 with a discussion of some other topics in the computer experiment literature, including design and emulator diagnostics.

Throughout this chapter, and indeed throughout this thesis, we treat uncertainty as a subjective property of one's beliefs unless otherwise stated. Probabilities and other measures of uncertainty are to be treated as subjective statements of belief and not inherent and measurable properties of the systems, models and concepts we shall introduce. For a detailed development of subjective probability theory, see, for

example, De Finetti [32].

2.1 On the use of computer models describing physical systems

Technological advances with respect to the memory capacity and speed of modern computers, have revolutionized the way we learn about complex systems. Physical processes that are difficult to measure or even impossible to experiment with, can now be investigated using computer models. These models represent physical processes through a series of mathematical equations, which are then solved using some complex numerical method.

Computer models are used in engineering to aid the design of mechanical systems such as liquid rocket injectors [92], automobile pistons [104], electrical circuits [8], and fusion capsules for spacecraft [35]. Simulators of much larger physical processes are also studied to help scientists better understand the world. Examples include aspects of climate such as the meridional overturning current [4], the thermosphere-ionosphere [97], and even the formation of galaxies in the beginning of the Universe [41].

The increasing importance of the computer model for scientific study led to the birth of a new subject, *the Design and Analysis of Computer Experiments* (DACE), popularised in the landmark paper by Sacks et al. [101]. Over the last 20 years the subject has gained increasing importance and interest from the wider scientific community, as scientists seek to better understand their models and the complex systems they represent [106].

Broadly speaking, there are three goals a scientist may have when using a computer model to learn about a complex system. These are

1. Improved knowledge about the physical processes.

2. Optimizing a particular set of design variables.

3. Decision support for policy makers.

In the Galaxy formation experiments, for example, the scientists wish to use the Galaxy simulator GALFORM [12] to better understand the Big Bang and the way the Universe was formed. This is an example of a goal of the first type. An example of the second is the automobile piston. A model of the piston is built, and the goal is to find the optimal shape with respect to certain constraints on wear and motion. As an example of a policy support problem, we might consider how best to reduce carbon emissions and how much to invest in appropriate technologies in order to "most favourably influence" the rate of the meridional overturning current.

Although policy problems may seem similar to optimization problems, we make them distinct here for a number of reasons. A policy problem involves attempting to influence the evolution of a complex system in time. An optimization problem however, seeks to minimize current output of a system with respect to some loss function. As such, it is often possible to obtain real-world observations of a system at particular settings to aid a decision maker in an optimization problem. One cannot test policy effects in the same way because there can only ever be one realisation of the future of the system. In that sense, policy decisions permanently change systems and their potential effects can only be explored using computer simulators. This makes our treatment of such problems more akin to that for type 1 goals than those of type 2.

Policy problems are the focus of this thesis. However we present methodologies traditionally used in addressing the other goals here. Application of these methods in order to provide policy support, as well as an overview of current methods in that area, will be presented in chapter 3.

2.1.1 Inputs and outputs of the computer code

Throughout this thesis, we refer to the output of a computer simulator as the vector-valued function $f(\cdot)$. There are three types of input to a computer simulator described in detail in chapter 2 of "The Design and Analysis of Computer Experiments" by Santner et al. [103]. These are *control variables, environmental variables* and *model variables*.

Control variables, henceforth to be referred to as *decision parameters*, we define

as those variables that we can change in order to affect the complex system and hence the model. These represent the policies that can be made and whose effect on the physical system we intend to investigate through the computer model. In the example for designing a liquid rocket injector, the decision parameters included areas for the tubes carrying oxygen and hydrogen, as well as the angle of hydrogen flow. The computer model and, if built, the fuel injector would behave differently for any choice of these variables. Decision parameters are denoted by the vector θ throughout this thesis.

Model variables, or model parameters, represent the inputs to the computer model that drive the physics and the boundary conditions in order to produce a solution. For example, Craig et al. [19] describe a simulator for an oil reservoir with 40 model inputs. Each are "multipliers" designed to fix permeability of rock in a particular region of the reservoir, or to describe the rate at which oil is allowed to flow through a particular fault. It is usually hoped that a certain setting of the model inputs makes the model behave like the physical system it represents (e.g. like the real oil reservoir described in the example). We discuss this assumption in further detail later. Model inputs are denoted by the vector x.

The third kind of input are what Santner et al. describe as "environmental variables." These inputs affect the output for a specific user. The example in the book is one of a model for a hip prosthesis, where stress on the implant is specific to the patient and their activity. We do not treat environmental variables in this thesis specifically and as such refer to all simulators of interest henceforth as functions with model inputs and decision parameters only. We denote the computer model with model inputs and decision inputs, $f(x, \theta)$.

2.1.2 Relating models and systems

Our interest in a particular computer model lies in how we may use it to learn about the complex physical system it aims to mimic. The general approach to this learning is to express our uncertainties about the system through the computer model, and then use computer experiments to update those uncertainties. Here, and throughout this thesis, we refer to the state vector of the complex system of

interest under decision θ as $y(\theta)$.

Define X to be the space of possible model inputs. If we believe that there is some input, $x^* \in X$ say, for which our simulator is "correct" and perfectly describes the system, we may write

$$y(\theta) = f(x^*, \theta), \tag{2.1}$$

and concentrate our energies on finding x^*. Unless the physical system is completely understood and the model absolutely perfect, this scenario is unlikely. For any application of sufficient complexity (complex enough to be interesting), the old adage "all models are wrong" will apply.

Computer models are often imperfect because they either do not contain all of the appropriate physics, or perhaps the physics are not completely understood. Mathematical expressions with simplifying assumptions may be used, and numerical solvers often approximate intractible analytical expressions. That said, a computer model still represents a serious attempt at describing the complex system and will generally be informative for it.

We extend (2.1) then, via the *best input* approach (c.f. Kennedy and O'Hagan [58], Higdon et al. [44], Bayarri et al. [9], Craig et al. [17]), i.e.

$$y(\theta) = f(x^*, \theta) + \eta(\theta), \tag{2.2}$$

where we define $\eta(\theta)$ to be the *model discrepancy* at θ. We specify that $\eta(\theta)$ is a stochastic process and is independent of the best input x^*, and of $f(x, \theta)$ for any x and θ. The model discrepancy (also known as the model inadequacy or bias function) represents physics that the model does not capture, as well as any features of the system we do not understand or have not anticipated. This assumption states that $f(x^*, \theta)$ is sufficient for the computer simulator, i.e. if we were to run the model at x^*, we could learn nothing more about the system from the model.

Interpretation and implications of the best input approach

The best input x^* does not necessarily correspond to a true, real-world, physical quantity. Model inputs representing known physical constants (e.g. acceleration due to gravity) may not have their "best" value matching their real-world value.

It may be helpful to think of relating the model and the system as an exercise in statistical fitting based on physical considerations, rather than a matching of physical and model quantities. In this respect, $f(x^*, \theta)$ represents the 'best fit' to the system, and x^* is the location in X that provides this. A detailed discussion of these ideas is presented in Kennedy and O'Hagan [58].

Assumption (2.2) defines $f(x, \theta)$ to be what Goldstein and Rougier 2004 [38] call a *direct simulator.* Specifying that $f(x, \theta)$ is a direct simulator represents a very strong belief statement. It implies a belief structure on similar models of the same system, and on any future improved versions of the model that may, or may not yet, be conceived of by the modellers. For example, suppose there exists, or will exist in the future, an improved version of the model with the same inputs, $f^{'}(x, \theta)$. As $f^{'}$ represents an improvement on f, there must be some input, x_0^* say, at which we can evaluate $f^{'}$ to resolve part of the discrepancy. We could then decompose the discrepancy as

$$\eta(\theta) = f^{'}(x_0^*, \theta) - f(x^*, \theta) + \eta^{'}(\theta), \tag{2.3}$$

where $\eta^{'}(\theta)$ is a process in θ. Therefore, (2.2) implies that $f^{'}(x_0^*, \theta) - f(x^*, \theta)$ represents further structural modelling of $\eta(\theta)$.

The best input assumption specifies $x^* \perp \eta(\theta)$ for all θ, which because of (2.3) may now look counter-intuitive. It is therefore dangerous to apply the best input assumption to a model when it is known that a better model exists, will be built or even can be conceived of, without considering its implications on our beliefs regarding improved models. We discuss these issues in much greater depth in Chapter 5, where we present a framework for providing policy support using models that are continually improving. For now it is enough to present an analysis using the best input assumption with the implication that $f(x, \theta)$ is our best simulator and that we won't have access to any other at any time in the future.

Whilst the best input approach may appear to be controversial, it is a very common treatment (perhaps more common is not to consider a discrepancy between the model and the system at all), and can be seen as analogous to traditional statistical models where the response is modelled via a regression plus error.

Equation (2.2) implies that learning about the location of x^* and about the

model's behaviour there, is our ultimate goal when using a computer simulator to learn about a physical system. In this section we describe methods for learning about the computer model's behaviour globally. We postpone formal treatment of x^* and of learning about the system to section 2.3.2.

2.2 Emulators

2.2.1 Justifying a stochastic approach

The computer models we describe are deterministic functions of their inputs. This means that running the model twice, at the same input values, will produce identical outputs.

For any real-world application involving a computer model, understanding local and global behaviour in the simulator is vital. For example, consider a model with only decision inputs $f(\theta)$, and suppose the 'best' decision, θ^*, was that which minimised $f(\theta)$. In order to locate θ^* using some numerical minimization routine, $f(\theta)$ must be evaluated many times. For the problems we are interested in, either very long run-times or high dimensional input spaces make this infeasible.

An *emulator*, often called a *surrogate* or *meta-model*, is a simple model that mimics the behaviour of the computer model. One idea in building an emulator (a process called *emulation*), is to use the computer model to construct a good fit to the simulator, and then use the emulator (which should be cheap to evaluate) in place of the model. For example, a sensible way to find the minimum θ^* we described earlier may be to build a good emulator and find the minimum of that. Another popular goal in emulation is to use the emulator as a description of our beliefs about the output of a computer model for some choice of inputs before the model has been run there.

Forrester and Keane [33] describe a number of emulation techniques based on the principles of numerical model fitting. Techniques mentioned include the fitting of polynomials, fitting of radial basis function, moving least squares and others. As computer models are deterministic, it may seem intuitive to consider emulation as a numerical analysis problem where the object is to fit a model that minimizes global

numerical error. Another popular approach however, is to model the emulator as a stochastic process or uncertain function.

The concept behind this derives from the idea that we want to be able to *predict* the output of the computer code for some x_0, θ_0. Without running the code we cannot be sure what the value of $f(x_0, \theta_0)$ will be, and this uncertainty is modelled stochastically. Referring back to (2.2), our goal is to provide a description of uncertainty in $f(x, \theta)$, combined with uncertainty about x^* and the discrepancy, in order to make inference about the complex system. This description of uncertainty will be most naturally served through the stochastic modelling of $f(x, \theta)$.

2.2.2 A general form of emulator

Whilst there are many different approaches to emulation, fit for many different purposes, the general form of the model remains consistent throughout much of the computer experiment literature. The emulator has general form

$$f(x, \theta) = \sum_{j=1}^{m} \beta_j g_j(x, \theta) + u(x, \theta), \tag{2.4}$$

where the $g_j(x, \theta)$ are (generally known) basis functions in x and θ, the β_j are unknown coefficients, and $u(x, \theta)$ is a mean-zero stationary process. Behaviour of this process is driven by its covariance function which, because of the stationarity, is a function of the distance between any two points. Normally this takes the form

$$Cov\left[f(x_1, \theta_1), f(x_2, \theta_2)\right] = \Sigma R(|(x_1, \theta_1) - (x_2, \theta_2)|, \Psi) \tag{2.5}$$

where Σ is a variance matrix for the model outputs and $R(|(x_1, \theta_1) - (x_2, \theta_2)|, \Psi)$ is a function of the distance between inputs and a vector of correlation parameters Ψ. For example, if we specified a Gaussian covariance function, then

$$R(|(x_1, \theta_1) - (x_2, \theta_2)|, \Psi) = \exp\{-((x_1, \theta_1) - (x_2, \theta_2))\Psi((x_1, \theta_1) - (x_2, \theta_2))^T\}.$$

Other correlation functions, including a more general exponential function and the Màtern, are described in Koehler and Owen [60]. A wider discussion of correlation functions for spatial processes in the area of Geostatistics is available in Diggle and Ribeiro Jr. [27].

Starting from this general form, we devote the rest of this section to presenting a number of the most popular methods of emulation. Each of these methods aims to use model runs (or training data) $F = (f(x_1, \theta_1), \ldots, f(x_n, \theta_n))$, to obtain a version of (2.4) fit for its intended purpose.

2.2.3 Kriging

In what follows we drop the distinction between input types to simplify the notation, because the methods we describe do not require it. We therefore write the simulator as $f(z)$ and further simplify the description by considering $f(z)$ as a scalar valued function for this section only. Note that this implies a scalar variance on the model outputs, which we denote σ, replacing the variance matrix Σ.

Sacks et al. [101] presented the frequentist approach to emulation known as *kriging*, in the case where the parameters Ψ, of the correlation function $R(|z_1 - z_2|, \Psi)$ are known. Kriging is an approach already used in other areas of statistics, particularly spatial statistics (see, for example, Cressie [20]), and we present here a few of the kriging methods as applied to computer experiments.

Generally speaking, the process of kriging involves finding the Best Linear Unbiased Predictor (BLUP) for data F. Assuming the form of the emulator as in (2.4), a linear predictor of $f(z)$ using data F is

$$\hat{f}(z) = c(z)F,$$

which is random in F. The BLUP is obtained by choosing $c(z)$ to minimise mean squared error

$$E\left[c(z)F - f(z)\right]^2,$$

subject to the unbiassedness constraint

$$E\left[c(z)f(z)\right] = E\left[f(z)\right].$$

Letting V denote the covariance matrix for $(u(z_1), \ldots, u(z_n))$, $v(z)$ the covariance between $u(z)$ and $(u(z_1), \ldots, u(z_n))$, and G the matrix $(g(z_1), \ldots, g(z_n))$, then it can be shown that the BLUP is

$$\hat{f}(z) = \hat{\beta}g(z) + v(z)V^{-1}(F - \hat{\beta}G),$$

where $\hat{\beta}$ is the usual least squares estimate $(G^T V^{-1} G)^{-1} G^T V^{-1} F$. This form of kriging, with known non-trivial basis functions in vector $g(z)$, is now known as *universal kriging.*

Modelling the computer code via a stochastic process in this way is seen, by the kriging community, as so powerful that the regression terms are usually replaced by a single mean [51]. Denoting this mean μ, equation (2.4) becomes

$$f(z) = \mu + u(z), \tag{2.6}$$

and using the BLUP to fit this emulator is called *Ordinary Kriging.*

Fitting the correlation parameters

One of the princple difficulties in kriging is estimating the values of Ψ, σ and μ (or β) (see, for example, Sacks, Schiller and Welch 1989 [100]). The goal within this methodology is to find good estimates to these values, and then use the BLUP and its standard error as the emulator for the model. There are two main ways these estimates are obtained; these are maximum likelihood estimation and cross-validation.

Maximum likelihood methods within the emulation literature require the assumption that $u(x)$ is Gaussian (e.g. Welch et al. [109]), so that F is multivariate normal. Using (2.6) and maximising the log-likelihood gives the expressions

$$\hat{\mu}(\Psi) = \frac{IV^{-1}F}{IV^{-1}I} \tag{2.7}$$

and

$$\hat{\sigma}^2(\Psi) = \frac{(F - I\hat{\mu}(\Psi))^T V^{-1} (F - I\hat{\mu}(\Psi))}{n}, \tag{2.8}$$

where I is an n-vector of 1's. Equations (2.7) and (2.8) are then substituted back into the likelihood, and the resulting "concentrated log-likelihood" is a function of Ψ and the data only. This is then maximised to find $\hat{\Psi}$, and then $\hat{\mu}(\hat{\Psi})$ and $\hat{\sigma}^2(\hat{\Psi})$ are computed from (2.7) and (2.8). Further details of these calculations can be found in Jones 2001 [50] and elsewhere.

Cross validation, often seen as an attractive method because it does not require further assumptions, involves iteratively leaving each data point out in turn and

re-fitting the emulator using the remaining $n-1$ points. Each new emulator is then used to predict the left out data, and the errors are computed and summed over all n data points. The required parameters can then be estimated by choosing the set that minimises this error. How simple this is in practice will depend on the magnitude of n and the number of individual correlation parameters in Ψ. More details of these methods can be found in Martin and Simpson [72].

Blind Kriging

The method of *Blind Kriging* (Joseph et al. [53]) takes the opposite stance on the power of ordinary kriging, seeking to use the unviersal kriging model with unknown basis functions $g(z)$. The general idea behind the approach is first to fit the basis functions using some well established statistical fitting method, and then proceed with kriging as normal. In the paper by Joseph et al. a Bayesian forward selection method was used to determine the basis functions. More details of the implementation of blind kriging, as well as a performance comparison with other kriging methods can be found in [53].

Application of kriging technology

Emulators built using kriging methods are very popular in engineering. Generally, these applications seek to find the optimum of the computer code by building a good emulator and finding its minimum. The building of the emulator may require an iterative approach using model evaluations at and around the minimum of the emulator to improve the fit. The standard errors for each prediction may often be used to aid design (see section 2.4.1) yet, in most respects, the emulator is treated as a function rather than a belief statement.

We now present the more natural Bayesian approach to emulation that allows us to express beliefs about the output of the computer model at any given location of its inputs.

2.2.4 Bayesian emulation

The frequentist philosophy is that there exists an underlying process from which the computer model is a draw. Runs of the model are then used to fit statistical models which we have called emulators, whose function in a classical setting is to be informative for the underlying process. This philosophy is clearly invalid when the code itself is deterministic, which calls into question what the emulator actually represents. In contrast, the Bayesian philosophy regards the emulator as a belief statement expressing our uncertainty regarding the code output for any value of the inputs. Runs on the simulator update these prior assessments, and this 'tuned' model combines all available data and any expert knowledge to form our current state of uncertainty regarding the computer model.

Learning about the computer model within this natural framework was first outlined by Currin et al. [24]. In their paper, the prior process was Gaussian with constant mean, and the parameters were determined via the maximum likelihood methods we met in section 2.2.3. Haylock and O'Hagan [43] give a more formal treatment to the Bayesian problem by including parameter uncertainty on β and σ^2. Following a Bayesian update of

$$f(x)|\beta, \sigma^2 \sim N(\beta g(x), \sigma^2 R(\cdot, \Psi))$$

by runs F, and given the prior form $p(\beta, \sigma^2) \propto \sigma^{-2}$, the parameters are integrated out of the joint posterior for $f(x), \beta, \sigma^2|F$, yielding a student t-distribution for $f(x)|F$.

An alternative prior form for $p(\beta, \sigma^2)$ is the normal inverse gamma, and this again yields a t-distibution in the posterior. This has been applied in the computer experiment literature (see, for example, Oakley and O'Hagan [82] or Rougier et al. [99]), and full details of the Normal Inverse Gamma update can be found in O'Hagan [84]. Note that these methods retain the assumption that correlation parameters Ψ are known.

A more formal treatment, considering uncertainty on the parameters Ψ in the context of a Gaussian process prior, is given in Kennedy and O'Hagan [58] where it is concluded that, regardless of any prior assumptions on these parameters, the full Bayes analysis handling this uncertainty will generally be infeasible. Their solution

is to fit $\Psi = \hat{\Psi}$ using code output, and to condition the posterior analysis on this estimate.

The Gaussian process emulation techniques we have described are relatively simple in their implementation because the posterior emulator has a closed form. The difficulty, as with most emulation techniques, lies in estimating or fitting Ψ and cross validation may be used to determine this. An issue with the use of the GP model is that strong prior judgements are required in order to obtain the closed form posteriors. This would be OK if we truly believed those judgements, or in certain cases if a very large amount of model data were available. Typically, relative to the dimension of the input space, our available data will be sparse anyway and hence our prior judgements may influence any posterior inferences quite heavily. In that sense then, if the GP prior assumptions are not truly believed, one must question the interpretability of the posterior.

There are two ways around this if we wish to adopt an approach that propagates our true uncertainties through the analysis. The first involves carefully stating or eliciting our prior beliefs in the form of probability distributions and then performing a large scale MCMC calculation. The second involves a partial prior specification and adopting the Bayes Linear approach we describe here.

2.2.5 Bayes Linear Emulation

Introduction to Bayes Linear Methods

It is traditional to consider uncertainties measured by probability. The Bayesian emulation methods we have so far described represent an attempt to express our uncertainty about the output of the computer code at a given point via a probability distribution. However, subjective uncertainty is a measure of our own state of knowledge and need not always be expressed or thought of in terms of probabilities.

Bayes Linear methods aim to express and manage uncertainty using expectation, and not probability, as a primitive quantity. Expectation is defined in the same way as De Finetti [32] defines prevision, namely as a 'fair price.' For example, suppose X is a random quantity and consider a ticket that pays X, then $E[X]$ is the fair price

you would be willing to pay for the ticket. Note that probabilities may be derived as the expectations of indictor functions. In using expectation as the primitive quantity, prior judgements such as prior expectation for a quantity may be considered directly, without first considering all possible probabilities for that quantity.

A full development of Bayes Linear methods, including philosophies regarding the use of expectation as primitive can be found in the book by Goldstein and Wooff [42]. We present here a few of the definitions that will be most useful to us and refer interested readers to the book for more details and a wider development.

Bayes Linear methods require, at the minimum, prior specification of expectations, variances and covariances across a collection of quantities of interest. To borrow the notation of Goldstein and Wooff [42] for a moment, suppose this collection is C. If we will observe $D = \{D_1, \ldots, D_k\} \in C$ then our beliefs about all of the other quantities in C will change. We model this learning through linear fitting. Consider our beliefs about $B = \{B_1, \ldots, B_r\} \in C$ given the values of D. Our *adjusted expectation* for B given D, written $E_D[B]$ is defined to be the linear combination that minimises

$$E\left[\left(B - \sum_{i=0}^{k} h_i D_i\right)^2\right]$$

over all collections $(h_0, h_1, \ldots, h_k)$ where $D_0 = 1$.

Definition 2.2.1 The **adjusted expectation** of collection B, given observation of collection D is

$$E_D[B] = E[B] + Cov[B, D]\, Var[D]^{-1} (D - E[D]), \tag{2.9}$$

where we use the Moore-Penrose generalised inverse if $Var[D]$ is singular.

Definition 2.2.2 The **adjusted variance** of B given D, written $Var_D[B]$ is defined as

$$\begin{aligned} Var_D[B] &= E\left[(B - E_D[B])^2\right] \\ &= Var[B] - Cov[B, D]\, Var[D]^{-1}\, Cov[D, B]. \end{aligned} \tag{2.10}$$

Definition 2.2.3 The variance of B resolved by D, $RVar_D[B]$ is defined as

$$RVar_D[B] = Var[E_D[B]] = Cov[B, D]\, Var[D]^{-1}\, Cov[D, B]. \tag{2.11}$$

Definition 2.2.4 The adjusted covariance between quantities B_1 and B_2 is

$$\begin{aligned} Cov_D\left[B_1, B_2\right] &= E\left[\left(B_1 - E_D\left[B_1\right]\right)\left(B_2 - E_D\left[B_2\right]\right)\right] \\ &= Cov\left[B_1, B_2\right] - Cov\left[B_1, D\right] Var\left[D\right]^{-1} Cov\left[D, B_2\right]. \end{aligned} \tag{2.12}$$

Interpretation of adjusted beliefs

Bayes Linear adjusted expectations can be viewed in one of three ways:

1. An approximation to the full Bayes posterior expectation; useful if the full calculation is too expensive or time consuming. The approximation is exact in certain cases, for example when C is multivariate normal.

2. An estimator for B obtained in an intuitive manner from reasoned prior judgements and real-world observations.

3. A natural generalization of conditional expectation, where the usual restriction that we must condition on indicator functions for a partition is dropped (see page 59 of Goldstein and Wooff for details).

Whilst it is the belief of this author that 3 is the correct interpretation, particularly in the case where a partial prior specification is all that can reasonably be expected, it may often be useful to keep the first interpretation in our minds. Problems concerning multivariate emulators for computer models are notoriously computer intensive and even without subscribing to the idea of adjusted expectation as a generalised conditional expectation, the methods we outline will still be important for policy problems where pragmatic approximations to difficult calculations will be required.

The adjusted variance does not depend on the data and is therefore not a true conditional variance (generalized or otherwise). It can be interpreted as a primitive prior measure of the residual variance in B once variance accounted for by D has been removed. For information concerning use of D to learn about the variance of B directly, see chapter 8 of Goldstein and Wooff.

Second Order Emulators

Bayes Linear methods allow us to make a partial prior specification for our emulator and to use model runs to update these judgements. Being able to make and use a partial prior specification will be important whenever making a fully probabilistic prior is particularly difficult or inappropriate. For example, if the computer model has high dimensional input and output spaces.

A second order emulator for a computer model uses the same form of stochastic process model (shown in (2.4)) as other emulators we have described. Instead of probability distributions being placed on the parameters β and Σ as they would be in a fully Bayesian emulation; means, variances and covariances are specified for each of the random quantities in the model. More precisely, we specify a mean and variance for the coefficient matrix β, and we specify the prior variance matrix Σ and form of the correlation function for $u(x, \theta)$. Values for the correlation parameters Ψ are fixed at some best choice $\hat{\Psi}$ as they would be if using the GP methods we described in section 2.2.4. We specify that the prior covariance between elements of β and of $u(x, \theta)$ is 0.

We now show how Bayes Linear methods are used to update a second order emulator with data obtained through running the model.

Updating the emulator: Notation

Suppose we obtain n runs of the model at $(x_1, \theta_1), \ldots, (x_n, \theta_n)$. Let F be the matrix whose columns contain these runs. Let $G = (g(x_1, \theta_1), \ldots, g(x_n, \theta_n))$ and $U = (u(x_1, \theta_1), \ldots, u(x_n, \theta_n))$ so that

$$F = \beta G + U$$

and F_{ij} is the ith output value of $f(x_j, \theta_j)$.

The quantity β is a matrix and so $Var\left[\beta\right]$ is treated as an array with four dimensions. We write

$$Var\left[\beta\right]_{ijkl} = Cov\left[\beta_{ij}, \beta_{kl}\right],$$

and similarly for the variance of any other matrix quantity. Other types of quantity we shall have to deal with involve computing the covariance between a vector and

a matrix, e.g. $Cov\left[u(x,\theta),U\right]$. In these cases we treat the object as an array with three subscripts. For the example we would write

$$Cov\left[u_i(x,\theta),U_{jk}\right] = Cov\left[u(x,\theta),U\right]_{ijk}.$$

Throughout this thesis, unless otherwise specified, we use Einstein's summation convention for repeated subscripts. For array objects A, B we write $A_{ijkl}B_{klmn}$ to mean

$$\sum_k \sum_l A_{ijkl}B_{klmn}.$$

The resulting object in this case would be $(AB)_{ijmn}$. Note that this convention only applies to repeated subscripts on different arrays that are multiplied together. In particular then

$$A_{ijkl}B_{klmm} = \sum_k \sum_l A_{ijkl}B_{klmm},$$

and repeated subscripts either side of addition signs are treated as normal. For the reader it may be useful to consider subscripts on array objects as dimension labels. A repeated subscript on adjacent arrays under this interpretation leads to the collapsing of this dimension in the resulting expression.

Updating the emulator: Calculations

The Bayes Linear adjustment of the second order emulator requires calculation of the expectation and variance of the simulator adjusted by the model runs F. We start by computing $E_F\left[f(x,\theta)\right]$ for some (x,θ), which by (2.9) is

$$E_F\left[f(x,\theta)\right] = E\left[f(x,\theta)\right] + Cov\left[f(x,\theta),F\right]Var\left[F\right]^{-1}\left(F - E\left[F\right]\right).$$

Let $\Delta = F - E\left[F\right]$, then

$$\begin{aligned}\Delta_{ij} = F_{ij} - E\left[F\right]_{ij} &= F_{ij} - E\left[\beta G + U\right]_{ij} \\ &= F_{ij} - E\left[\beta\right]_{im} G_{mj}.\end{aligned}$$

We have

$$\begin{aligned}Var\left[F\right]_{ijkl} &= Var\left[\beta G\right]_{ijkl} + Var\left[U\right]_{ijkl} \\ &= Cov\left[(\beta G)_{ij},(\beta G)_{kl}\right] + Var\left[U\right]_{ijkl} \\ &= Cov\left[\beta_{im}G_{mj},\beta_{kn}G_{nl}\right] + Var\left[U\right]_{ijkl} \\ &= Var\left[\beta\right]_{imkn} G_{mj}G_{nl} + Var\left[U\right]_{ijkl}, \qquad (2.13)\end{aligned}$$

where $Var\left[U\right]_{ijkl} = Cov\left[u_i((x_j,\theta_j)), u_k((x_l,\theta_l))\right]$ is computed using (2.5).

By the linearity of adjusted expectations we have

$$E_F\left[f(x,\theta)\right] = E_F\left[\beta g(x,\theta)\right] + E_F\left[u(x,\theta)\right], \tag{2.14}$$

where

$$\begin{aligned} E_F\left[\beta g(x,\theta)\right]_i &= E\left[\beta_{im} g_m(x,\theta)\right] + Cov\left[\beta g(x,\theta), \beta G\right]_{ikl} Var\left[F\right]^{-1}_{klop} \Delta_{op} \\ &= E\left[\beta\right]_{im} g_m(x,\theta) + Cov\left[\beta, \beta G\right]_{imkl} Var\left[F\right]^{-1}_{klop} \Delta_{op} g_m(x,\theta) \\ &= E_F\left[\beta\right]_{im} g_m(x,\theta), \end{aligned}$$

and

$$E_F\left[u(x,\theta)\right]_i = Cov\left[u(x,\theta), U\right]_{ikl} Var\left[F\right]^{-1}_{klmn} \Delta_{mn}.$$

Note that to update the regression surface we need only update the coefficient matrix which does not depend on (x,θ).

Using (2.10), the adjusted variance of the simulator at (x,θ) is

$$\begin{aligned} Var_F\left[f(x,\theta)\right] &= Var\left[f(x,\theta)\right] - Cov\left[f(x,\theta), F\right] Var\left[F\right]^{-1} Cov\left[F, f(x,\theta)\right] \\ &= Var\left[\beta g(x,\theta)\right] - Cov\left[\beta g(x,\theta), F\right] Var\left[F\right]^{-1} Cov\left[F, \beta g(x,\theta)\right] \\ &\quad + Var\left[u(x,\theta)\right] - Cov\left[u(x,\theta), F\right] Var\left[F\right]^{-1} Cov\left[F, u(x,\theta)\right] \\ &\quad - Cov\left[\beta g(x,\theta), F\right] Var\left[F\right]^{-1} Cov\left[F, u(x,\theta)\right] \\ &\quad - Cov\left[u(x,\theta), F\right] Var\left[F\right]^{-1} Cov\left[F, \beta g(x,\theta)\right] \\ &= Var_F\left[\beta g(x,\theta)\right] + Var_F\left[u(x,\theta)\right] \\ &\quad + Cov_F\left[\beta g(x,\theta), u(x,\theta)\right] + Cov_F\left[u(x,\theta), \beta g(x,\theta)\right]. \end{aligned} \tag{2.15}$$

We write

$$\begin{aligned} Var\left[\beta g(x,\theta)\right]_{ij} &= Cov\left[(\beta g(x,\theta))_i, (\beta g(x,\theta))_j\right] \\ &= Var\left[\beta\right]_{imjn} g_m(x,\theta) g_n(x,\theta), \end{aligned}$$

so that the adjusted variance of the regression surface is then

$$\begin{aligned} Var_F\left[\beta g(x,\theta)\right]_{ij} &= Var\left[\beta\right]_{iqjp} g_q(x,\theta) g_p(x,\theta) \\ &\quad - Cov\left[\beta g(x,\theta), \beta G\right]_{ikl} Var\left[F\right]^{-1}_{klmn} Cov\left[\beta G, \beta g(x,\theta)\right]_{mnj} \\ &= Var_F\left[\beta\right]_{iqjp} g_q(x,\theta) g_p(x,\theta). \end{aligned}$$

For completeness, the remaining elements of (2.15) are

$$Var_F\left[u(x,\theta)\right]_{ij} = Var\left[u(x,\theta)\right]_{ij} - Cov\left[u(x,\theta),U\right]_{ikl} Var\left[F\right]^{-1}_{klmn} Cov\left[U,u(x,\theta)\right]_{mnj}$$
$$Cov_F\left[\beta g(x,\theta),u(x,\theta)\right]_{ij} = -Var\left[\beta\right]_{iqkr} G_{rl} g_q(x,\theta) Var\left[F\right]^{-1}_{klmn} Cov\left[U,u(x,\theta)\right]_{mnj}$$
$$Cov_F\left[u(x,\theta),\beta g(x,\theta)\right]_{ij} = -Cov\left[u(x,\theta),U\right]_{ikl} Var\left[F\right]^{-1}_{klmn} Var\left[\beta\right]_{msjp} G_{sn} g_p(x,\theta).$$

Note that these last two covariances are the transpose of one another.

2.2.6 Practical emulation for large-scale models

The kind of models used for studying complex physical systems to aid policy judgements pose particular challenges. Large numbers of different model inputs combined with a number of decisions can lead to high dimensional model input spaces. The model output too may be very complex, containing multiple types of quantity represented at many locations in time and space.

Building an accurate emulator in the form of (2.4) in such a high dimensional setting may be infeasible given limited access to model runs and expert judgements. Even if this is not the case, the number of inputs and outputs may still cause computational problems, so that updating via model runs or even evaluating the emulator becomes almost as time consuming as running the model. A number of methods have been suggested in the computer experiment literature to help circumnavigate these issues and we discuss some of them here.

Screening

The aim of screening is to use model runs to identify output variables that appear insensitive to changes in the model inputs. Another way to consider screening is the attempt to identify a manageable subset of model outputs that tell most (if not all) of the important features of the story. Examples of useful screening methods can be found in Cumming and Wooff [23] and in Saltelli [102].

Active variables

Often it will be the case that only a small subset of the model inputs drive most of the variability in the output. By emulating the model as a function of its active

inputs plus a residual in its inactive inputs, the scale of the emulation problem can be dramatically reduced. Let $x^A \in x$ be the subset of active model inputs, then we re-write (2.4) as

$$f(x,\theta) = \sum_{j=1}^{m} \beta_j g_j(x^A,\theta) + \xi(x^A,\theta) + \delta(x,\theta), \tag{2.16}$$

where the residual $u(x,\theta)$ has been decomposed into a stationary process $\xi(x^A,\theta)$ in the active variables, and a 'nugget' residual $\delta(x,\theta)$ in the remaining variables. This practical form of emulator seen, for example, in Cumming and Goldstein [21] and Craig et al. [19], requires substantial expert judgement or many model runs to help identify the active variables.

Other dimension reduction techniques in emulation include principal component methods and application of neural networks. A review of these is available from Maniyar et al. [69].

The effect of decision variables

For many complex physical systems, policies considered could dramatically effect the evolution of the system. If the computer model is a good one, these effects will be present in the model and hence should be captured by our emulator.

Consider, for example, a climate model that outputs both annual mean UK temperature, and the flow of the meridionnal overturning current. In addition to a collection of model inputs, suppose the simulator includes CO_2 emissions as a decision input. If we wish to emulate the UK temperature, the stochastic model we have described may be impractical. This is because evolution of temperature as a function of emissions may be very different depending on whether or not the model MOC collapses. If a particular choice of emissions induces MOC collapse, the model becomes fundamentally different (as does the system it mimics). There may be new active variables, the regression surface we would fit may be different and the residual surface may have different properties.

A more flexible version of (2.4), with the ability to capture large scale behaviour

changes in the model is

$$f(x,\theta) = \sum_{j=1}^{m} \beta_j(\theta) g_j(x^A(\theta),\theta) + u^\theta(x,\theta), \tag{2.17}$$

where the coefficient matrix β, the active variables x^A, and the form of the residual process all depend on θ.

How one would fit such an emulator must be highly model specific. Generally though it will be infeasible to model $\beta(\theta)$, $x^A(\theta)$ and the form of the covariance function for $u^\theta(x,\theta)$ as continuously changing variables in θ. A practical approach would be to identify areas of the decision space in which the model behaves consistently and fit the usual model (2.4) within each of these areas. In the example discussed above, we may identify areas of the decision space that induce MOC collapse and effectively fit 2 emulators. One for UK temperatures under θ when MOC has collapsed, and one for when it has not. The model emulator in that case would become a mixture of emulators.

Although building emulators of complex models for policy problems is a difficult and important subject, it remains an unsolved problem that is beyond the focus of this thesis. The policy support methods we outline later will presume an emulator in the usual form is built for the computer model. However we believe that the methods will be easily adapted to more complex emulators if the application requires it.

Practical fitting

In a Bayesian or Bayes Linear emulation, updating our prior via model runs is computationally demanding. In fact if we have q outputs and n model runs, the update is an $O((nq)^3)$ calculation. This means that as either the number of outputs grows, or as we obtain more model runs, the cost of emulation quickly spirals out of control. Alternative emulation techniques have been suggested by An and Owen [3] and Rougier [95], in order to overcome these problems. Rougier [96] shows how, in a Bayesian setting, one can exploit separability in the emulator to make the update an $O(nq)$ calculation.

Whichever method of emulation we decide to use, choosing the basis functions, correlation function, and making prior judgements is hard. A popular way to fit

the emulator is to use existing model runs to help build a picture of the model. Often however, the computer model is so expensive that very few of these runs are available. One technique for emulator fitting, often available in these cases, is *multi-level emulation.* The idea being that a cheap, fast version of the model is run many times to inform us about our expensive simulator. Multi-level emulation methods are central to our methods of policy support and we devote a whole section to reviewing and explaining them in chapter 3.

We now turn our attention away from the construction of emulators and focus on how we may use them to learn about the physical systems we're interested in. Throughout this next section, our emulators are treated as a model of our uncertainty regarding the output of the simulator at unobserved inputs. Usually our emulators will be second order, but we refer to the fully probabilistic versions when appropriate.

2.3 Using emulators to study physical systems

In section 2.1.2 we introduced the best input assumption (2.2) linking the output of the computer simulator to reality. By expressing our uncertainties for the discrepancy $\eta(\theta)$ and the best input x^*, a model emulator completes our uncertainty specification for the complex system $y(\theta)$. In this section we show how to combine the emulator with observations of the physical system in order to learn about x^* and $y(\theta)$.

2.3.1 History Matching

The best input assumption (2.2) states that we can obtain all information about the complex system contained in the model by evaluating it at x^*. Attempting to locate x^* then, is a naturally popular goal when performing computer experiments.

Having built an emulator for the computer model, the general approach is to use observations of the real-world system combined with the emulator to locate possible matches. We denote the real-world data z_{t_0} observed with error via

$$z_{t_0} = y_{t_0} + e_{t_0}, \tag{2.18}$$

where e_{t_0} is observational error and the subscript t_0 indicates 'historical' system values. Note that under this interpretation z_{t_0} does not depend on θ. This will be true in cases where y is seen as a constantly evolving system (such as climate), and θ is a decision or policy that has not been made yet. For other kinds of system we may be able to obtain decision-dependent data. For example, if considering safety features of a car, we might be able to build a car with a particular safety configuration and test it. As we are mainly interested in computer models for policy problems, we assume that our data represents historical system values and hence does not depend on θ.

History matching is an important methodology in the computer experiment literature with a number of different applications. The general idea behind the method is to identify areas of the model input space that contain possible values of x^* and a Bayes Linear approach to the method was introduced in the important paper by Craig et al. [18]. Using an emulator for the model, the technique locates those areas of the input space which might, if we were to evaluate the model there, replicate the real-world data (modulo the error structure imposed by the variance of the observation error and the discrepancy).

History matching can inform model builders as to whether or not their model is able to replicate the system at all. For example, it is possible that careful history matching reveals no areas of the input space containing a plausible x^*. Understanding whether or not a model is informative for the system it was built to mimic is a key goal when performing computer experiments.

A history match will often lead to our ruling out areas of the model input space as containing no plausible matches. This is extremely useful if we are able to obtain further runs of the model, as it allows us to re-focus our emulators. By re-sampling the model and emulating only on those areas of the input space thought to contain possible matches, we can build emulators that are more informative for the complex system. History matching and re-focussing can be done many times until we are satisfied that our emulator is as accurate around the best input as we can make it. This last application is a convincing argument for history matching to be performed as a first step, whatever the method of emulation used and regardless of the ultimate

goal of the analysis.

History matching uses the model emulator, the observed system data and a measure of implausibility to determine whether or not a particular choice of model inputs x could be a value of x^*. The standard implausibility measure $I(x)$, based on the malhalanobis distance between $f_{t_0}(x)$ and z_{t_0}, is

$$I(x) = (E\left[f_{t_0}(x)\right] - z_{t_0})^T Var\left[E\left[f_{t_0}(x)\right] - z_{t_0}\right]^{-1}(E\left[f_{t_0}(x)\right] - z_{t_0}).$$

Writing

$$E\left[f_{t_0}(x)\right] - z_{t_0} = E\left[f_{t_0}(x)\right] - f_{t_0}(x) + f_{t_0}(x) - y_{t_0} + y_{t_0} - z_{t_0},$$

we arrive at the more intuitive form

$$I(x) = (E\left[f_{t_0}(x)\right] - z_{t_0})^T (Var\left[f_{t_0}(x)\right] + Var\left[\eta_{t_0}\right] + Var\left[e_{t_0}\right])^{-1}(E\left[f_{t_0}(x)\right] - z_{t_0}), \tag{2.19}$$

if $x = x^*$.

The expectations and variances in (2.19) indicate our current beliefs about the model and so if we have observed a set of runs, these are our adjusted moments. Large values of the implausibility measure indicate poor matches. To define 'large' in this case, a tolerance level may be selected by the experimenter so that values larger than the tolerance are rejected as plausible matches. Discussion regarding how this tolerance might be chosen and how implausibility may be visualised, as well as alternative measures of implausibility can be see in Craig et al. 1996 [18], Craig et al 1997 [19] and Cumming and Goldstein [21].

2.3.2 Forecasting

Of principle interest to policy makers, is predicting the future behaviour of a complex system under a given strategy. By specifying beliefs about x^*, the model emulator in combination with judgements regarding the discrepancy completely determines our beliefs about $y(\theta)$ for any θ through (2.2).

Depending on our approach to emulation, forecasting methodology may be quite varied. In a Gaussian Process setting, for example, forecasting follows naturally from

calibration, which involves formally learning about x^* from z_{t_0}. Assuming a Gaussian process discrepancy and a Gaussian process emulator as described in section 2.2.4, closed form expressions may be obtained for $p(x^*|F, z_{t_0})$ and moments of the posterior Gaussian process $y(\theta)|x^*, F, z_{t_0}$. Moments of $y(\theta)|F, z_{t_0}$ may be obtained by integrating moments of $y(\theta)|x^*, F, z_{t_0}$ with respect to $p(x^*|F, z_{t_0})$ numerically. Values of the distribution function for $y(\theta)|F, z_{t_0}$ may be obtained in a similar way.

A fully Bayesian MCMC forecasting calculation would be extremely challenging, both numerically and in the prior specification. Handling the relationship between F, Ψ, x^* and z_{t_0} for any θ, for example, will be particularly difficult.

If we have made the philosophical or pragmatic choice to use Bayes Linear methods and to construct a second order emulator, the forecast method is not as intuitively straightforward. Assuming a prior probability distribution on the best input, $p(x^*)$, it is unclear how the system data z_{t_0} should inform us about x^*.

From (2.2) and (2.18) we have

$$z_{t_0} = f_{t_0}(x^*) + \eta_{t_0} + e_{t_0},$$

so z_{t_0} must contain some information about x^*. However, to update our distribution for x^* using z_{t_0} we require $p(z_{t_0}|x^*)$ which is undefined by our second order specification.

There is also a philosophical question to be addressed here. Learning, within the Bayes Linear paradigm, is facilitated by linear fitting. There is limited information contained in z_{t_0}, and our principle interest when forecasting is in the value of $y(\theta)$. Therefore it could be argued that it is more appropriate to adjust beliefs about $y(\theta)$ by z_{t_0}, rather than waste time and resources extracting the potentially limited information regarding x^* contained in z_{t_0} first. Whether or not this argument is valid will depend on the model, the application and the beliefs of the analyst.

We stated previously that a history match should be performed regardless of the ultimate goal of the analysis. Assuming we have used z_{t_0} already to perform a history match, our model emulators should be accurate around plausible locations of x^*. We may then consider that we have extracted and handled the information about x^* contained in z_{t_0} in an appropriate way. If we accepted this, we would be left free to adjust $y(\theta)$ by z_{t_0} directly. Our approach to forecasting then is to perform

two, separate, Bayes Linear fits. The first adjusts the emulator by the model runs and allows us to construct a covariance between past and future values of the system. The second adjusts these system beliefs by real-world data.

Before moving on to describe the method of forecasting in detail, we note that in the case where we still have access to further runs of the computer model, inserting a calibration step and obtaining model runs at the expected values of x^* can yield powerful forecasts. These ideas and a methodology applying to policy problems are developed in Chapter 6. The forecasting we describe here assumes that we have exhausted our access to the computer model and must combine our emulators with real-world observations to express beliefs about future states of the complex system under different policy strategies.

Deriving system beliefs

We outline here the Bayes Linear forecasting methodology introduced by Craig et al. in [17] and apply it to models with decision inputs. We then introduce a number of novel approaches and techniques designed to make the calculations manageable.

As we are using Bayes Linear methods to learn about the complex system, the ultimate goal is to obtain a mean and variance for $y(\theta)$ using our emulator. Once second order beliefs about the complex system have been derived, we may adjust them using the data z_{t_0} to obtain a forecast. Using (2.2) we write

$$\begin{aligned} E\left[y(\theta)\right] &= E\left[E\left[f(x^*,\theta)|x^*\right]\right] + E\left[\eta(\theta)\right] \\ &= \int_X E\left[f(x^*,\theta)|x^*\right] p(x^*)dx^*, \end{aligned} \tag{2.20}$$

so that the decision-dependent expectation of the complex system is obtained by integrating the expected emulator over the prior distribution for x^*. As we stated above, we are assuming that our emulators have already been tuned via model runs so that $E\left[f(x^*,\theta)|x^*\right]$ is actually the expectation of the emulator given x^*, adjusted by runs F through equation (2.14).

We may express the variance of the system for given θ as

$$\begin{aligned} Var\left[y(\theta)\right] &= Var\left[f(x^*,\theta)\right] + Var\left[\eta(\theta)\right] \\ &= Var\left[E\left[f(x^*,\theta)|x^*\right]\right] + E\left[Var\left[f(x^*,\theta)|x^*\right]\right] + Var\left[\eta(\theta)\right], \\ &= \int_X Var\left[f(x^*,\theta)\right] p(x^*)dx^* + Var\left[\eta(\theta)\right] \\ &\quad + \int_X E\left[f(x^*,\theta)\right] E\left[f(x^*,\theta)\right]^T p(x^*)dx^* - E\left[y(\theta)\right] E\left[y(\theta)\right]^T, \end{aligned} \tag{2.21}$$

where again the moments of the emulator are adjusted by runs F. We now show, using index notation, how each of these integrals are computed as functions of the components of the emulator and the model runs. The derived expressions will then be used to illustrate the principle computational difficulty encountered when forecasting, and to develop forecasting approaches that exploit certain structures within the emulator that exist or that we are willing to impose. For notational convenience we consider the moments of the emulator within the integrands of the following expressions to be implicitly conditioned on x^*.

Combining (2.20) and (2.14) we have

$$E\left[y(\theta)\right] = \int_X E_F\left[\beta g(x^*,\theta)\right] p(x^*)dx^* + \int_X E_F\left[u(x^*,\theta)\right] p(x^*)dx^*, \tag{2.22}$$

where

$$\begin{aligned} \left[\int_X E_F\left[\beta g(x^*,\theta)\right] p(x^*)dx^*\right]_i &= \int_X E_F\left[\beta\right]_{im} g_m(x^*,\theta)p(x^*)dx^* \\ &= E_F\left[\beta\right]_{im} \left[\int_X g(x^*,\theta)p(x^*)dx^*\right]_m, \end{aligned}$$

and

$$\left[\int_X E_F\left[u(x^*,\theta)\right] p(x^*)dx^*\right]_i = \left[\int_X Cov\left[u(x^*,\theta),U\right] p(x^*)dx^*\right]_{ikl} Var\left[F\right]^{-1}_{klmn} \Delta_{mn}.$$

Define

$$W(x,\theta)_{ijk} = Cov\left[u(x,\theta),U\right]_{ijk}$$

then using (2.15), we have

$$\begin{aligned}\int_X Var_F\left[f(x^*,\theta)\right]p(x^*)dx^* &= \int_X Var_F\left[\beta g(x^*,\theta)\right]p(x^*)dx^* \\ &+ \int_X Var_F\left[u(x^*,\theta)\right]p(x^*)dx^* \\ &+ \int_X Cov_F\left[\beta g(x^*,\theta), u(x^*,\theta)\right]p(x^*)dx^* \\ &+ \int_X Cov_F\left[u(x^*,\theta), \beta g(x^*,\theta)\right]p(x^*)dx^*, \qquad (2.23)\end{aligned}$$

where

$$\begin{aligned}\int_X Var_F\left[\beta g(x^*,\theta)\right]_{ij}p(x^*)dx^* &= Var_F\left[\beta\right]_{iqjp}\int_X g_q(x^*,\theta)g_p(x^*,\theta)p(x^*)dx^* \\ &= Var_F\left[\beta\right]_{iqjp}\left[\int_X g(x^*,\theta)g(x^*,\theta)^T p(x^*)dx^*\right]_{qp},\end{aligned}$$

and

$$\begin{aligned}\int_X Var_F\left[u(x^*,\theta)\right]_{ij}p(x^*)dx^* &= Var\left[u(x^*,\theta)\right]_{ij} \\ &- \int_X W(x^*,\theta)_{ikl}Var\left[F\right]^{-1}_{klmn}W(x^*,\theta)_{jmn}p(x^*)dx^* \\ &= Var\left[u(x^*,\theta)\right]_{ij} \\ &- Var\left[F\right]^{-1}_{klmn}\left[\int_X W(x^*,\theta)W(x^*,\theta)p(x^*)dx^*\right]_{ikljmn}.\end{aligned}$$

The ijth element of the final two integrals in (2.23) is

$$Var\left[\beta\right]_{imkn}G_{nl}Var\left[F\right]^{-1}_{klop}\left[\int_X g(x^*,\theta)Cov\left[u(x^*,\theta),U\right]p(x^*)dx^*\right]_{mjop}$$

and

$$\left[\int_X g(x^*,\theta)Cov\left[u(x,\theta),U\right]p(x^*)dx^*\right]_{nikl}Var\left[F\right]^{-1}_{klop}Var\left[\beta\right]_{omjn}G_{mp}$$

respectively. To complete the variance calculation we require

$$\int_X E_F\left[f(x^*,\theta)\right]E_F\left[f(x^*,\theta)\right]^T p(x^*)dx^*,$$

whose ijth element is

$$\begin{aligned}&E_F\left[\beta\right]_{im}E_F\left[\beta\right]_{jn}\left[\int_X g(x^*,\theta)g(x^*,\theta)^T p(x^*)dx^*\right]_{mn} + \\ &E_F\left[\beta\right]_{im}\left[\int_X g(x^*,\theta)W(x^*,\theta)p(x^*)dx^*\right]_{mjkl}Var\left[F\right]^{-1}_{klnp}\Delta_{np} + \\ &E_F\left[\beta\right]_{jm}\left[\int_X g(x^*,\theta)W(x^*,\theta)p(x^*)dx^*\right]_{mikl}Var\left[F\right]^{-1}_{klnp}\Delta_{np} + \\ &Var\left[F\right]^{-1}_{klmn}\Delta_{mn}\left[\int_X W(x^*,\theta)W(x^*,\theta)p(x^*)dx^*\right]_{ikljpq}Var\left[F\right]^{-1}_{pqrs}\Delta_{rs}. \qquad (2.24)\end{aligned}$$

Having obtained $E[y(\theta)]$ and $Var[y(\theta)]$ by computing (2.20) and (2.21) through (2.22), (2.23) and (2.24), we complete the Bayes Linear forecast by adjusting these moments by z_{t_0}. From (2.9) and (2.10) these adjustments are

$$E_{z_{t_0}}[y(\theta)] = E[y(\theta)] + Cov[y(\theta), z_{t_0}] Var[z_{t_0}]^{-1} (z_{t_0} - E[z_{t_0}]) \tag{2.25}$$

and

$$Var_{z_{t_0}}[y(\theta)] = Var[y(\theta)] - Cov[y(\theta), z_{t_0}] Var[z_{t_0}]^{-1} Cov[z_{t_0}, y(\theta)], \tag{2.26}$$

where from (2.18) we see that

$$Var[z_{t_0}] = Var[e_{t_0}] + Var[y_{t_0}(\theta)].$$

The required covariances in (2.25) and (2.26) are obtained from $Var[y(\theta)]$ in a similar fashion.

The Computational Burden

We have shown that integrating out the best input to obtain judgements about $y(\theta)$ requires computation of 5 integrals. We must integrate $g(x^*, \theta)$, $Cov[u(x^*, \theta), U]$, and the outer products of these 2 quantities with each other and with themselves, with respect to $p(x^*)$. Generally, these integrals may not be obtained analytically and must be evaluated numerically. In appendix B we present a special case where uniform $p(x^*)$ combined with specific regressors and a Gaussian correlation function lead to closed form expressions for each integral.

Integrating functions of the regressors only is relatively straight forward numerically. This is because regressors are usually simple functions like monomials. Integrals containing $Cov[u(x, \theta), U]$, however, will take significantly longer. This is because for every numerical integration point x, the covariance array must be constructed by calculating covariances between $u(x, \theta)$ and the residuals from the model runs. If x (or the subset of x chosen as active variables) is high dimensional, or we have a large collection of runs, or even a complicated and expensive correlation function, constructing and storing the array may be time consuming. That it must be rebuilt at every numerical integration point provides the most significant challenge when obtaining a forecast.

Although we have established that integrating out the best input to obtain a forecast is challenging, its difficultly should be judged relative to the cost of building and running the simulator. Using powerful state of the art computers, as we would when running the simulator, we should expect to be able to perform these integrations. This argument is convincing when the goal is simply to predict a one-off future: for example, if there were no decision inputs and we only wanted to understand how a system is likely to evolve as in Craig et al. [17], or if a policy had been chosen already and we wanted to explore its likely impact on the system. Our application, on the other hand, is providing decision support for policy makers. To aid decision making, the very least we must be able to do is produce forecasts for a large number of different policies so that the decision maker can compare their performance. The techniques we outline in chapter 3 require many forecasts to provide powerful decision support tools.

We are motivated then, by a need to make many forecasts, to develop methods of forecasting quickly. Our focus will not be on the numerical integration rule itself, but on exploiting potential structural features of the emulator.

2.3.3 Separability

As discussed, the major issue in integrating out the best input will be that the expensive calculations must be performed for any θ we require a forecast for, and that in providing decision support we will require many of these forecasts. There are structures that may exist, or that we may choose to impose upon $g(x, \theta)$ and $R(|(x, \theta) - (x', \theta')|, \Psi)$, that remove the dependence of θ from the numerical integrations altogether. Should they be naturally present in our emulators or should we be willing to impose them, the issue we describe is completely resolved.

In what follows we use the operator '$*$' to indicate elementwise multiplication of two objects in the appropriately labelled dimension; i.e. for matrix objects M, N, the operation $M_{ij} * N_{ij}$ returns the product of the ijth element of M and the ijth element of N. This is distinguished from $M_{ij}N_{ij}$ which we have defined already to

be

$$\sum_i \sum_j \left[M_{ij} * N_{ij} \right].$$

We first consider separability in $g(x, \theta)$. Suppose we may write

$$g(x, \theta) = k(\theta) * h(x),$$

where $h(x)$ and $k(\theta)$ are vectors of the same length as $g(x, \theta)$. Then

$$\left[\int_X g(x^*, \theta) p(x^*) dx^* \right]_m = k_m(\theta) * \left[\int_X h(x^*) p(x^*) dx^* \right]_m \tag{2.27}$$

and

$$\begin{aligned} \int_X g_q(x^*, \theta) g_p(x^*, \theta) p(x^*) dx^* &= \int_X (k_q(\theta) * h_q(x^*))(k_p(\theta) * h_p(x^*)) p(x^*) dx^* \\ &= k_q(\theta) k_p(\theta) * \int_X h_q(x^*) h_p(x^*) p(x^*) dx^* \\ &= \left[k(\theta) k(\theta)^T \right]_{qp} * \left[\int_X h(x^*) h(x^*) p(x^*) dx^* \right]_{qp}. \end{aligned} \tag{2.28}$$

The integrals in (2.27) and (2.28) have no dependence on decision θ and may be computed once and stored. In stipulating separability in $g(x, \theta)$ then, we allow cheap evaluations of

$$\int_X g(x^*, \theta) p(x^*) dx^* \quad \& \quad \int_X g(x^*, \theta) g(x^*, \theta)^T p(x^*) dx^*,$$

following a one-time evaluation of

$$\int_X h(x^*) p(x^*) dx^* \quad \& \quad \int_X h(x^*) h(x^*)^T p(x^*) dx^*,$$

for any chosen θ. We now consider separability in the residual process.

Suppose we specify that the residual process $u(x, \theta)$ had a covariance function that was separable in x and θ, so that

$$Cov\left[u_i(x, \theta), u_j(x', \theta') \right] = \Sigma_{ij} r^x(|x - x'|) r^\theta(|\theta - \theta'|).$$

Let Ω be the matrix whose columns contain the design in the model inputs used to compute F and similarly let Θ be the design in the decision inputs. Denoting the ith design point (Ω_i, Θ_i), then

$$\left[\int_{-\infty}^{\infty} W(x^*, \theta) p(x^*) dx^* \right]_{ikl} = \Sigma_{ik} r^\theta(|\theta - \Theta_l|) * \left[\int_X r^x(|x^* - \Omega_l|) p(x^*) dx^* \right] \tag{2.29}$$

and

$$\left[\int_X W(x^*,\theta)W(x^*,\theta)p(x^*)dx^*\right]_{ikljmn} = \Sigma_{ik}\Sigma_{mj}r^\theta(|\theta-\Theta_l|)r^\theta(|\theta-\Theta_n|) * \int_X r^x(|x^*-\Omega_l|)r^x(|x^*-\Omega_n|)p(x^*)dx^*. \tag{2.30}$$

We can therefore compute the integrals

$$\int_X r^x(|x^*-\Omega|)p(x^*)dx^* \quad \& \quad \int_X r^x(|x^*-\Omega|)r^x(|x^*-\Omega|)^T p(x^*)dx^*$$

once and then find (2.29) and (2.30) for any θ via a relatively cheap calculation.

If we have both separability in $g(x,\theta)$ and in the residual process, we may make similar simplifications to the final numerical integral required,

$$\left[\int_X g(x^*,\theta)W(x^*,\theta)p(x^*)dx^*\right]_{mjop}.$$

This can be written as

$$\begin{aligned}
&\int_X (k_m(\theta) * h_m(x^*))\Sigma_{jo}(r^\theta(|\theta-\Theta_p|) * r^x(|x^*-\Omega_p|))p(x^*)dx^* \\
&= \Sigma_{jo}k_m(\theta)r^\theta(|\theta-\Theta_p|) * \int_X h_m(x^*)r^x(|x^*-\Omega_p|)p(x^*)dx^* \\
&= \Sigma_{jo}k_m(\theta)r^\theta(|\theta-\Theta_p|) * \left[\int_X h(x^*)r^x(|x^*-\Omega|)p(x^*)dx^*\right]_{mp}.
\end{aligned} \tag{2.31}$$

If imposed, these two mathematical structures make obtaining decision-dependent forecasts cheap, provided initial integration of the required $x-$dependent quantities has been done.

It may be that one or both of these separability assumptions does not represent our true beliefs about the computer model. If our emulators are not separable everywhere in the ways we have demonstrated above, the element-wise nature of the calculation of the integrals means that we may still use separability locally. For example, our vector of regressors may contain terms such as $sin(x_i\theta_j)$, making $g(x,\theta)$ inseparable. However, there may be elements of $g(x,\theta)$ that can be separated and whenever possible these should be identified and used to reduce the number of θ-dependent numerical integrations in a forecast.

Having located and exploited local separability in emulators that are not fully separable, we may still be left with very expensive θ-dependent numerical integrations which are infeasible to obtain more than a handful of times. In that case, we

may be forced into looking for other ways in which the forecasting calculations can be simplified. The different ideas we describe now are all variations on a theme we call *fast forecasting*.

2.3.4 Fast Forecasting

We define a *fast forecast* to be a forecast that approximates the careful forecast we might make, given enough time and computer power, in a fraction of the time it would normally take. We view fast forecasts as being informative for our careful, expensive forecasts. The ability to provide quick and reasonable approximations to careful forecasts is central to the methods of decision support we describe in chapter 3.

The most obvious fast forecasting technique we may consider is to use a less accurate but faster numerical integration method. How we choose to handle numerical integration, both for careful and fast forecasting, will be an important, problem-specific, consideration. The fast forecasting ideas we present in this section aim to exploit the structure of our emulators and assume that the numerical methods of integration have been decided.

Having exploited any element-wise separability in $g(x, \theta)$, we should have only a handful of decision-dependent integrations involving just the regressors. Generally, regressors are simple, cheap functions to evaluate and, relative to constructing and storing functions of $Cov\left[u(x, \theta), U\right]$, they are trivial to integrate. Our focus when developing fast forecasts then, is on how the integrals involving $Cov\left[u(x, \theta), U\right]$ may be simplified.

If, in building our emulator, a lot of effort was spent on finding a good global regression so that the variance of the residual process is small, the contribution of any integrals involving $u(x, \theta)$ may also be considered small or even negligible. Additional weight may be added to this argument if the points used to evaluate U are spread out and if the correlation lengths in our emulator are small. In these

cases a good fast approximation to the forecast may be obtained by setting

$$\left[\int_X W(x^*,\theta)p(x^*)dx^*\right]_{ijk} = 0,$$

$$\left[\int_X W(x^*,\theta)W(x^*,\theta)p(x^*)dx^*\right]_{ijklmn} = 0,$$

and

$$\left[\int_X g(x^*,\theta)W(x^*,\theta)p(x^*)dx^*\right]_{ijkl} = 0.$$

This type of fast forecast is so attractive because it removes all of the difficulty in constructing and storing the large covariance arrays. Essentially, we reduce the decision-dependent forecasting problem to one of evaluating a handful of inseparable, yet easy to compute, functions across our numerical integration scheme. It is perhaps, though, a little drastic to have gone to all the effort of carefully building the emulator and accounting for all of our uncertainties in the residual process, only to ignore their collective impact on our beliefs about the system.

There are two main applications where we would advocate using this type of forecast. The first is one in which a large number of very approximate forecasts are sufficient to give us an initial picture of how the forecast behaves in different areas of the decision space. Our intention would not be to use these forecasts directly in decision support, but to help build more sophisticated models that focus our search for 'good' forecasts. The second, and perhaps the more interesting application, is when θ is sufficiently far away from any of the design points that it alone takes $Cov\,[u(x,\theta),U]$ approximately to zero for any $x \in X$. If our design Θ is relatively sparse, this may be true for a substantial proportion of the decision space.

For norms that are monotonic in each of the elements of the vectors they measure, the distance of a given point (x,θ) from design point (Ω_i,Θ_i) is greater than the distance between (Ω_i,θ) and (Ω_i,Θ_i). Hence

$$Cov\,[u(x,\theta),u(\Omega_i,\Theta_i)] < Cov\,[u(\Omega_i,\theta),u(\Omega_i,\Theta_i)]$$

for $i = 1,\ldots,n$ and for all θ. For a given θ we can test if $Cov\,[u(x,\theta),U]$ is approximately zero for any $x \in X$ by fixing $x = \Omega_i$ (for some i), so that the contribution from x is at its maximum. We then compute $\max\{Cov\,[u(\Omega_i,\theta),U]\}$. If this value

is very close to zero, we may consider this type of fast forecast to be a good approximation to the careful forecast for the chosen θ.

If θ is close enough to any of the design points and we would prefer a more accurate fast forecast, there are other methods we may explore. One idea is to partition the decision space by considering the distance of any θ from its nearest design point. For one θ within each partition, we compute a careful forecast including each of the integrals involving $Cov\left[u(x,\theta), U\right]$. We then make the approximation that the contribution from these integrals is the same throughout each partition. Given a handful of initially expensive forecasts, then, under this scheme we would be able to compute decision-dependent forecasts almost as quickly as if we made the assumption that all contributions from the emulator residual were zero.

These fast forecasts may be useful when we expect most or all of the points in U to contribute to the covariance. If our data is sparse relative to the volume of the decision space or even if we have a short correlation length, it may be that only a handful of the points in U have any covariance with $u(x,\theta)$ for a given θ.

Suppose, as with a previous method, we fix $x = \Omega_i$ for some i. Instead of now focusing on the closest design point to θ, we now look for the smallest distance any point a can be from θ, such that we judge the residual correlation to be zero between points at θ and points at equivalent and greater distances. For any Θ_j further from θ than this distance, we may approximate the covariance between $u(x,\theta)$ and $u(\Omega_j,\Theta_j)$ by zero for all x.

More formally, let

$$d = \left\{\min(||\theta - a||) : Cov\left[u(\Omega_i,\theta), u(\Omega_i, a)\right] \approx 0\right\},$$

then for $j = 1, \ldots, n$, if $||\theta - \Theta_j|| \geq d$ we set

$$Cov\left[u(x,\theta), u(\Omega_j,\Theta_j)\right] = 0$$

for all x.

This fast forecasting method serves to reduce the computation of $Cov\left[u(x,\theta), U\right]$ to the evaluation of a handful of non-zero covariances. Although perhaps requiring more initial effort to set up than other fast forecasting techniques we have proposed,

this particular method should lead to substantial computational savings and very close approximations to our careful forecasts.

If we have a relatively small number of decisions to make, our design in the decision space may not be very sparse. This may lead us to conclude, based on any of the tests described so far, that we need to integrate most of $Cov\left[u(x,\theta),U\right]$ and the outer products involving it quite carefully. Often though, in such cases, X may by high dimensional so that our design points Ω are sparse in X.

For the norms we described earlier, we also have

$$Cov\left[u(x,\theta),u(\Omega_i,\Theta_i)\right] < Cov\left[u(x,\Theta_i),u(\Omega_i,\Theta_i)\right]$$

for $i=1,\ldots,n$ and for all x.

For each design point we may construct a neighbourhood so that for any x not contained in one of these n neighbourhoods, the covariance between $u(x,\theta)$ and U is approximately zero. In doing this we reduce the size of the numerical integration by limiting it to those neighbourhoods we have selected. For $i=1,\ldots,n$, let

$$\tau_i = \left\{\min(||x-\Omega_i||) : Cov\left[u(x,\Theta_i),u(\Omega_i,\Theta_i)\right] \approx 0\right\}.$$

Our required integrals then become

$$\left[\int_{\Gamma} W(x^*,\theta)p(x^*)dx^*\right]_{ijk},$$

$$\left[\int_{\Gamma} W(x^*,\theta)W(x^*,\theta)p(x^*)dx^*\right]_{ijklmn},$$

and

$$\left[\int_{\Gamma} g(x^*,\theta)W(x^*,\theta)p(x^*)dx^*\right]_{ijkl},$$

where

$$\Gamma = \bigcup_{i=1}^{n}\{x : ||x-\Omega_i|| < \tau_i\}.$$

Let

$$\Gamma_i = \{x : ||x-\Omega_i|| < \tau_i\},$$

then if $\bigcup_{i=1}^{n-1}\bigcup_{j>i}\{\Gamma_i \cap \Gamma_j\} = \emptyset$ we only require

$$\sum_{p=1}^{n}\left[\int_{\Gamma_p} W(x^*,\theta)p(x^*)dx^*\right]_{ijk},$$

$$\sum_{p=1}^{n}\left[\int_{\Gamma_p} W(x^*,\theta)W(x^*,\theta)p(x^*)dx^*\right]_{ijklmn},$$

and

$$\sum_{p=1}^{n} \left[\int_{\Gamma_p} g(x^*, \theta) W(x^*, \theta) p(x^*) dx^* \right]_{ijkl} .$$

The condition that none of the neighbourhoods intersect will often be satisfied if Ω is sparse in X and so seems a reasonable assumption.

By constructing these neighbourhoods, we simplify all numerical integrations involving the residual to the sum of n integrations over small neighbourhoods of the design points. The hope will then be that far fewer covariance arrays will be required in the numerical integrations so that the forecast calculation is significantly cheaper for any θ.

How much of the accuracy of a forecast we are willing to trade in for speed of the approximation will depend on the application. In chapter 3 we show how to combine fast forecasting and careful forecasting to aid decision support. We conclude the current chapter by mentioning other topics in the computer experiment literature, particularly design. Whilst our decision support methods have not exploited any of these additional topics, we feel it is important to include them for completeness and because they suggest interesting areas for further work.

2.4 Design and other topics

2.4.1 Design of computer experiments

A substantial proportion of research into using computer models to learn about complex systems is concerned with design. The issue of which values of x and θ to evaluate $f(x, \theta)$ for when only a limited number of runs are available is called the design problem.

There are two principle philosophies behind choosing a design. One is that the design points should cover as much of the input space as possible so that we can understand the global model behaviour, as well as we could hope to, across the entire input space. These designs are known as *space filling designs*. The other school of thought is that designs should be made for purpose and should depend on the way we wish to use the model. For example, our goal may be to build the most accurate

emulator possible and we would aim to choose our runs to minimise some property of the emulator variance, such as its trace. These designs are known as *criteria based designs*. We give a brief overview of the main techniques for both design types here, and point interested readers to chapters 5 and 6 of Santner et al [103] for a more detailed introduction.

Space filling designs

Perhaps the most popular class of space filling design is the Latin Hypercube. The aim of Latin Hypercube sampling is to ensure that each area of the input space is represented in the design. Suppose we wish to perform n runs of the computer model and that there are d dimensions in the input space. A Latin Hypercube design would divide the domain of each of these d inputs into n parts and produce an n point design that represents each of these parts exactly once.

If we have a distribution across our inputs, the division of each input dimension is done so that each of the n parts has equal probability. Dividing the input space in this way leads to a partition of n^d cells. Once a cell is selected as being in our sample, the design point within that cell is obtained by uniform sampling. One way of choosing cells to ensure that the design is a Latin Hypercube is to create an $n \times d$ matrix whose columns are randomly chosen permutations of $\{1, \ldots, n\}$. The ith row in this case represents the cell in which the ith design point is to be located.

A desirable property of space filling designs is that the points should not be too close together. Whilst Latin Hypercubes ensure that many different areas of the input space are covered by our design, they do not necessarily have this property. There are other space filling design methods that use criteria to ensure that the points are sufficiently far away from each other. For example, maximin designs choose points to maximise the minimum distance between any two points in the design.

By restricting the class of possible designs to Latin Hypercubes and applying some distance criterion, space filling designers appear to be able to get the best of both worlds. Designs of this type, including the maximin Latin Hypercube, are explored, for example by Morris and Mitchell [77] and Liefvendahl and Stocki [67].

An alternative space filling design method is the use of Sobol sequences (see the MUCM toolkit [2]). Sobol sequences are deterministic sequences that have 'quasi-random' properties and are computed using binary fractions. Sobol sequences have been used to evaluate integrals more accurately and efficiently than Monte Carlo methods (see, for example, Blatman et al. [11]), but their application to the design of computer experiments is relatively recent. For details on how the sequences are constructed see Press et al. [89]. An attractive feature of the Sobol sequence as a design method compared with Latin Hypercubes, is that Sobol sequences may be added to, so that we may contruct sequential space filling designs.

Criteria based designs

Space filling designs may often be adopted to provide an initial set of runs used to build or train emulators. If we already have an emulator, or are implementing a Bayesian emulation strategy with informative priors, then selecting the location of new runs via a space filling design may not be an optimal strategy. We may have some experimental goal to consider, such as locating an optimal θ or providing decision support. It may be that we have a limited budget and want our emulator to be as accurate as possible throughout the design space. We may be looking for plausible history matches or require the best possible forecast for a one-off θ. In each of these cases, different designs will lead to different results.

For each of the examples we have given we may 'cook up' a criterion that forms the basis of a good design for our purpose. For example, Craig et al. [17] suggest a design criterion so that our experiment gives us the most accurate forecasts we can expect. The idea is to choose the design that minimises the trace of the forecast variance, $Var\left[y(\theta)\right]$, which implicitly depends on the design through the model runs and the forecasting equations in section 2.3.2.

In the optimization literature, criteria based designs are particularly important. The goal of the experiment is to locate the θ that minimises the computer model. Sequential criteria based design, where the model is evaluated at each step and the emulators re-fitted, is an often used approach. A popular criterion is that of expected improvement. The design is chosen so that the expected distance between

the function at our current best θ and the new point is maximised. Details of these algorithms may be found in, for example, Williams et al. [111] or Huang et al. [47].

For a policy problem where only a limited number of runs can be obtained from the computer model, the location of these runs is very important. We discuss these issues in more depth at the end of Chapter 3.

2.4.2 Emulator Diagnostics

The methods we propose in chapter 3 will assume we have a good emulator of the computer model. Having built a model emulator, assessing how accurately it mimics the computer model and whether or not our prior beliefs were consistent with the observed runs is very important. Whilst emulator diagnostics are important, we limit ourselves to a brief discussion of them here as they are not the focus of this thesis.

One popular diagnostic is the 'leave one out' method. One run is left out of the collection of runs used to update a Bayesian emulator, and the left out point is predicted using this new, adjusted statistical model. We then measure the standardized distance from the observed to the predicted point. If our prediction is more than 2 or 3 standard deviations away from the observed value, our prior beliefs may not be reasonable and we may decide to re-examine them. This method is similar to the cross validation techniques we described earlier, although the parameters themselves are not re-fitted.

Bayes linear methods provide a suite of diagnostic tools that can be applied to second order emulators. These are presented in detail in Chapter 4 of Goldstein and Wooff [42]. Emulator diagnostics for Gaussian process emulators are discussed in Bastos and O'Hagan [7].

Chapter 3

Model aided decision support for policy problems with intervention

In this chapter we apply the theory for managing uncertainty in computer models and using them to learn about physical systems to complex policy problems. In section 3.1 we describe the types of policy problem in which we are interested. In section 3.2 we review some of the current practices used to provide decision support in these problems using Integrated Assessment. In section 3.3 we describe an idealized approach to solving policy problems with our most powerful computer models and the technologies described in chapter 2. We conclude that it is not possible to solve these problems exactly and propose a method of decision support that combines fast forecasting techniques introduced in 2.3.4 with more careful and expensive forecasts. In section 3.4 we review multi-level emulation techniques and discuss multi-level emulation of expected losses. In section 3.5 we describe our method of Sequential Emulation for gaining insight into the properties of the large decision trees defined by policy problems. In section 3.6 we present our Sequential Emulation algorithm. Section 3.7 presents some ideas for using Sequential Emulation to provide policy support. We offer a discussion in section 3.8.

Throughout this thesis the standard results in Bayesian decision theory are assumed known. For a detailed introduction and development of this theory, see, for example, DeGroot [26].

3.1 The policy problem with intervention

3.1.1 Loss models

Policy makers increasingly rely on computer models to aid policy judgements influencing complex systems (see, for example, Linkov and Burmistrov [68]). Not only must policy makers use computer models to understand the complex system and its reaction to different decisions, they must also use them to understand that system's impact on the world's population and economy, as well as the cost of executing a given policy. To motivate our discussion throughout this chapter it will be useful to have a real-world example in our minds. We use an example of a real-world policy problem in addressing climate change.

Governments throughout the world must make policy regarding carbon emissions and investment to tackle climate change. This is in order to avoid what the UNFCC defined as "dangerous anthropogenic interference with the climate system" [1]. Whilst most industrialised countries have committed themselves to reducing emissions to specific targets [59] by signing the Kyoto agreement [85], the policy makers must decide how these targets are to be achieved. To do this they must not only use models to learn how different emissions scenarios and investment programs may affect climate, but they must also use models to understand how different climate states and cuts to emissions will affect their economies and populations.

This second model we refer to as the *loss model.* A loss model's inputs will include the policies to be made, the state of the complex system under those policies, and a number of parameters designed to describe how populations and the economy evolve and the damage suffered due to different future states of the system, as well as any preferences of society and the decision makers. The output of a loss model should be in units of utility. This may involve computing loss as a monetary value and applying a social welfare function (we see an example of such a model in chapter 4). As the loss model involves mimicking the evolution of the economy over time, it may be as complex and expensive as the model for the system.

3.1.2 Optimal policy and intervention

The goal of the policy maker, if he believes his loss function truly expresses his preferences in units of utility, is to find the policy that minimizes his expected loss with respect to plausible future states of the system, where loss is negative utility. Often though, the loss model may be so complicated that the policy maker may not be certain that any parametrization gives a true reflection of his preferences in utility units. Therefore, policy makers will also be interested in policies that have relatively "safe" risk profiles and that appear robust to various plausible parametrizations of the loss model.

Once a policy is chosen and implemented, it will influence future states of the complex system. As the system evolves under a given policy, it may be observed and policy makers may wish to adapt their strategy. We call these adaptations *policy interventions* when they follow observation of the system. As an example, consider a policy made today in which we decide to do nothing to abate climate change. In a number of years, if the climatic impacts of global warming are steadily worsening, one would hope that policy makers would decide to intervene and cut emissions, rather than adhere to their original policy. A natural approach would be to make a number of periodic observations and interventions. A policy maker will usually intend their long term strategy to be like this before he makes any decision today.

The theory of sequential decision making (for a thorough development see DeGroot [26]) states that knowledge of future observations and opportunities to make decisions in the future should be accounted for when making decisions today. Therefore, the potential for future observation-based policy revision must be carefully accounted for in any decision support we may provide. We now formalize the policy problem we intend to study in this chapter.

3.1.3 Formalizing the policy problem

A policy maker must make a decision today, parametrised by θ_{t_0}, in order to favourably influence future states of a complex system. To aid him, he has observations of the system z_{t_0} (defined on page 25). At m different points in the future, $t_1, \ldots, t_m$,

the decision maker may observe the system under his current policy, and make new policy θ_{t_i}, for $i = 1, \ldots, m$. Let $\theta = (\theta_{t_0}, \theta_{t_1}, \ldots, \theta_{t_m})$, then the complex system is denoted $y(\theta)$. Observations of the system made from one policy intervention to the next at time t_i are

$$z_{t_i}(\theta_{t_0}, \theta_{t_1}, \ldots, \theta_{t_{i-1}}) = y_{t_i}(\theta_{t_0}, \theta_{t_1}, \ldots, \theta_{t_{i-1}}) + e_{t_i}.$$

We assume a finite number of interventions is made and write $y_{t_f}(\theta)$ to denote the future state of the system after all observations and interventions have been made. For clarity, t_f represents all future times, for which we have system information and well defined preferences over outcomes, after the final intervention has been made.

In order to inform him about the complex system, the policy maker may make runs on a computer model $f(x, \theta)$, whose outputs are linked to the system via the best input assumption (2.2). For now we consider that this model is fixed, however in Chapter 5 we will allow the model to be improved and updated by the time we wish to make interventions. The impact of any policy and system realization on the economy and the policy maker's preferences over different economic states is measured by the loss model $L(x_L, y(\theta), \theta)$, where x_L is a set of loss model parameters and the output of $L(\cdot)$ is in units of utility. We present an example of the influence diagram defined by this problem for $m = 2$ in figure 3.1.

Before we address the problem with a view to providing decision support, we give a brief review of some of the current methods used in policy making with computer models. Our experience of attempts to address these issues in the literature involves a particular class of simulators known as *Integrated Assessment Models*.

3.2 Integrated Assessment

Attempts to explore optimal policy often involve Integrated Assessment (IA) models (see, for example, van Asselt and Rotmans [107], Keller et al. [55]). These models couple an often simplified model of the complex system with a loss model that reacts and interacts with it to make one model. The IA model will have a policy as well as a number of model parameters as inputs and will output a utility. Changes in the

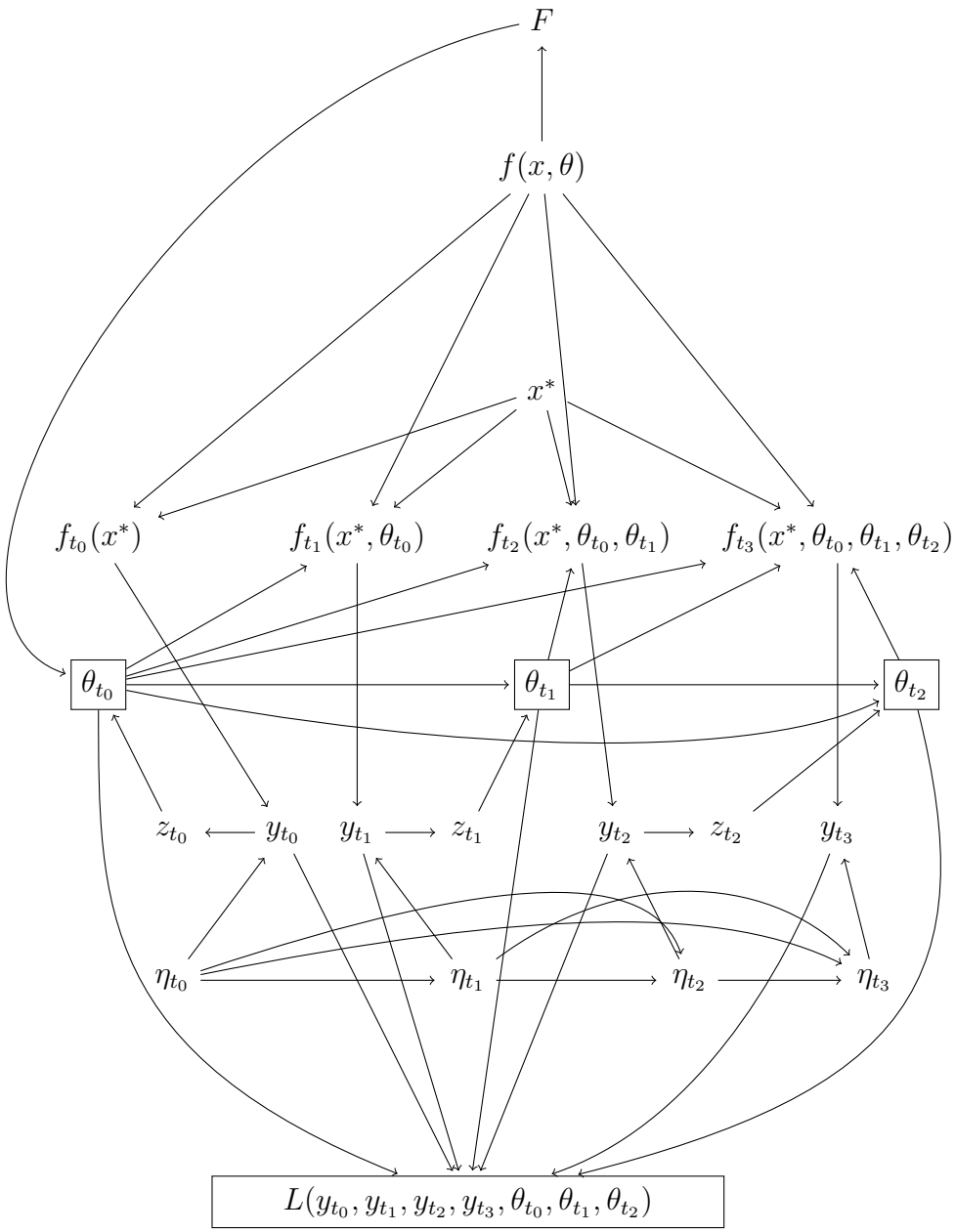

Figure 3.1: Our statement of the decision problem and the modelling statements made in chapter 2 define an influence diagram. This figure presents an example of this influence diagram for the case where we have 2 downstream intervention points at times t_1 and t_2.

model economy and the complex system interact at each time step, providing feedbacks and allowing adaptive policies to be considered and evaluated in the context of the model.

IA models are often seen as able to provide useful insight for policy makers that stand-alone, expert models (such as a powerful climate model) cannot [94]. Potential interactions between socio-economic output and the complex system are modelled explicitly (although simply), allowing the potential for policy makers to judge which interactions may be important. For example, some IA models for climate change use a simple climate model to compute temperature at each time step which is then fed into a 'damage function' describing the effects of that temperature on populations and the world economy at that time step. Due to the model temperatures and damages, there may be some required investment or migration that might, for example, alter the amount of land covered by vegetation. This would then feed back into the climate model and affect temperature at the next time step.

3.2.1 Locating optimal policy

One potential treatment of IA models is to consider them as a single model for utility of a policy for some complex system; for example, a model of utility for an emissions policy. Although this utility depends on a submodel for climate change, the entire function is treated as a 'black box' reflection of preferences. In such a case we can find the optimal policy, perhaps with respect to certain constraints. For example, McInerney and Keller [73] use a simple IA model to find an optimal emissions strategy subject to the constraint that the model MOC does not collapse.

Rather than search over all potential policies for the optimum, IA models are often used to compare the performance of a finite number of policies (see, for example, Budescu et al. [14]). In such cases, the 'optimal' policy may be regarded as being the best of the considered policies. Good policy support should explore a wide range of potential decisions and their impacts (Izrael and Semenov [49] give a discussion of this referring to emission strategies).

The 'optimal' policy, that being the decision that minimizes the loss output by the IA model, is only optimal with respect to a given parametrization of the model.

A useful method of policy support is known as *scenario analysis*, and aims to outline the different policy consequences of alternative views of the world. If using an IA model to find optimal policy, one may wish to find that optimum under different parameterizations of the policy maker's preferences. Keller et al. [56], for example, consider optimal CO_2 sequestration for a range of different costs per ton of carbon removal.

3.2.2 Investigating adaptive strategies

Policy strategies that are capable of adapting to changes in the real world are seen as better than one-off optimizations (see, for example, Lempert et al 2000 [66]). The idea is to write down a set of potential actions to take today, together with a list of potential actions one might take given different observations of the system in the future, and use the IA model to explore the quality of each potential strategy. Examples of such exploration can be found in Lempert et al. 1996 [65] and Bankes [5].

Note that the study of adaptive strategies does not (directly, at least) address the intervention problem outlined in section 3.1.3. Whilst potential future observations are considered, they are only ever treated inside the IA model. That is to say that observations of the real world do not drive the utility of an adaptive strategy. Instead it is driven by system related sub-model outcomes, treated as if they were observations of the real world.

3.2.3 Handling and reporting uncertainty

The IA approach aims to treat many sources of uncertainty in providing policy support. We have already discussed the use of scenario analysis in the context of seeing its effect on optimal policy. Scenario analysis can be a useful tool in decision support, particularly for policy problems. Experts often disagree on the subject of potential long-term behaviour of the system, the damage it may cause, or the preferences of society for different outcomes (see, for example, Morgan and Keith [75], Morgan et al. [76]). The goal in these cases is to find policies that are

robust to these different scenarios.

Uncertainty regarding the choice of model (with reference to particular equations for subcomponents rather than a completely different model) and what the IA modellers call 'parameter uncertainty' is often addressed (see, for example, Casman et al. [15]). Exploratory modelling (also known as computer assisted reasoning) allows the policy maker to explore an ensemble of models (or parametrizations of the model) over a number of adaptive strategies, in order to assist decision making (see Lempert [64]).

Kann and Weyant [54] describe a number of methods of uncertainty analysis in IA modelling, including a formal treatment of the sequential decision problem for interventions (although future "observations" are still model-based). They also discuss sensitivity analysis, and how parameter uncertainty may propagate through the model to give a distribution of potential losses. Appropriate methods of policy support advocated by Kann and Weyant, and others cited here, include providing a 'portfolio of actions' that perform well for a variety of different scenarios, and for any well studied policy, a measure of risk dispersion (a risk profile) for outcomes.

3.2.4 Drawbacks to the IA approach

The policy support techniques in IA modelling that we have described, such as computer assisted reasoning, rely on relatively fast run times. The simplifications made to the physics of the complex system, micro and macro economies, and the preferences of policy makers within IA models are therefore often substantial. As far as we have been able to see, however, the differences between the model or the individual sub-models, and the systems they represent, are hardly addressed.

The sub-model of the complex system is occasionally calibrated to real-world data (see, for example, Nordhaus and Boyer [80]), however, neither observation error nor model discrepancy is considered. It remains unclear whether a calibrated sub-model is of any more use in a policy setting than an un-calibrated one, if uncertainty regarding the model discrepancy is not formally treated. For example, consider an IA model for the emissions problem. The simplified climate model used to compute temperature may be calibrated to the temperature record. However, when testing

a policy where CO_2 emissions lead to concentrations never before observed in the real-world, is the calibrated climate model still informative for climate in this state? This question is very important if we are to use the model to advise policy makers regarding climate change.

Policy makers are often mistrustful of computer models and are concerned with the difference between model prediction and reality (see Brugnach et al. [13]). It seems a reasonable requirement, then, that any genuine policy support should carefully handle the difference between the model and the system. The problem with this, however, may be that having used a very simple submodel in an IA model, the additional uncertainty would dominate any analysis.

The IA community has begun to work on methods of using more powerful, expensive and accurate sub-models to build their simulators. Leimbach and Jaeger [63] suggest a framework for using powerful sub-models in IA called modules. They create extra models designed solely to enable the different modules to talk to each other (potentially translating different programming languages) and to run quickly and simultaneously. Application of the approach, known as Community Integrated Assessment Modelling (CIAM), can be seen in, for example, Leimbach et al. [62]. Although CIAM aims to use the most powerful expert models from each research community, it is difficult to see how this will ever scale up to the most powerful expert models whilst retaining the useful, relatively quick to evaluate, policy support properties desired by the practitioners of integrated assessment. As an example, consider the climate model at the Hadley Centre currently takes around a month to run on a super computer.

In the rest of this chapter we describe our approach to policy support. This approach allows the use of our most powerful models from each discipline, carefully addresses the difference between the model and the real world, and allows the potential for actual future observations and interventions to influence the support we offer. In chapter 5 we extend this approach to allow future updates to the model. We believe this approach is suitable and appropriate for the most challenging real-world policy problems. The methods are based on the statistical machinery we described in chapter 2 for managing uncertainty using computer models

3.3 The expected loss surface

Our approach to policy support is to combine state of the art models for complex systems with loss models and Bayesian emulation technology, in order to provide a visualisation of the policy maker's expected loss surface as a function of policy made today. Once obtained, this map of the loss surface can be used to provide many different types of decision support. We discuss decision support using the expected loss surface in section 3.7.

In what follows we treat the loss model as a known function $L(\theta, y(\theta))$. Doing this assumes that an x_L has been chosen that accurately reflects the policy maker's preferences and the behaviour of the economic system in response to the complex system. In reality we may be as uncertain regarding the 'true' parametrization of the loss model as we are regarding the location of x^* for our computer model. In spite of our primitive treatment of the loss function, we believe that our methodology is novel in that it shows how powerful expert models of the complex system may be used to provide policy support. We carefully handle the issue of system observations and policy intervention and have hopefully paved the way for extensions to be developed that treat the loss model more formally. In section 3.7, we discuss our own forms of scenario analysis that aim to address some of our uncertainties about the proper form of the loss model and in section 3.8, offer discussion regarding treating $L(x_L, y(\theta), \theta)$ as a computer model.

3.3.1 The ideal solution

In what follows it will be useful to develop some new notation to handle collections of decisions and observations. Firstly, we simplify our current notation by suppressing the dependence of z_{t_i} and y on the policies θ_{t_i} for $i = 0, \ldots, m$. We define

$$z^k = z_{t_0}, z_{t_1}, \ldots, z_{t_k}$$
$$\theta^k = \theta_{t_0}, \theta_{t_1}, \ldots, \theta_{t_k}$$

for $k = 1, \ldots, m$.

The policy problem defined in section 3.1.3 defines a decision tree. We present a snapshot of this decision tree for one observation and intervention ($m = 1$) in

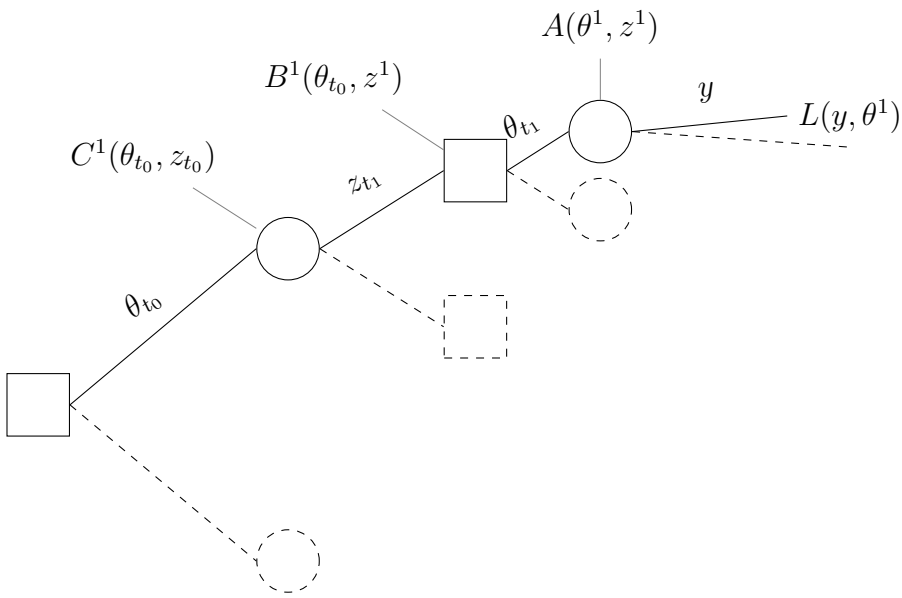

Figure 3.2: A snapshot of the decision tree defined by the intervention problem when $m = 1$. The node labels correspond to the quantities defined by (3.1), (3.2) and (3.3). The dotted lines represent the infinite number of alternative paths defined by the different decisions and observations we may make.

figure 3.2. The decision tree describes how policy is made in time. We choose θ_{t_0}, observe z_{t_1} and use the new information to choose θ_{t_1}. This pattern continues until the final observation z_{t_m} and intervention θ_{t_m} have been made, at which point we experience y_{t_f} and realize associated loss $L(y, \theta)$. In order to solve this decision tree we must recursively compute expectations at each of the circular chance nodes, and choose the decision to minimise these expectations at each of the adjacent square decision nodes. This method of rolling back the decision tree, known as backwards induction, we now describe mathematically for this particular problem.

Define

$$A(\theta^m, z^m) = \int_{-\infty}^{\infty} L(y, \theta^m) p(y|z^m, \theta^m) dy, \tag{3.1}$$

where $p(y|z^m, \theta^m)$ is a probability distribution for the state of the system given decisions θ^m and observations z^m. If this distribution accurately reflects our beliefs about the system under these conditions, then $A(\theta^m, z^m)$ is our expectation on the right-most chance node of the full decision tree for the branch defined by θ^m, z^m.

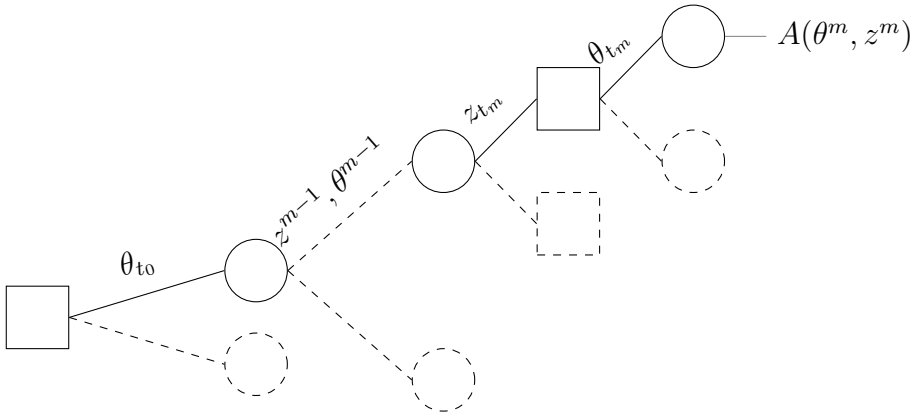

Figure 3.3: This image shows the decision tree for the full intervention problem after the expected losses as defined by (3.1) have been computed for each branch defined by θ^m and z^m. The dashed line between the two solid chance nodes represents the $m-1$ decisions and observations that are taken before we observe z_{t_m}.

The information at each of the right-most chance nodes, labelled $A(\theta^m, z^m)$ for any θ^m, z^m is now sufficient for all information contained in the loss function to the right of it. We may therefore focus our energies on the rolled back tree, a snapshot of which is depicted in figure 3.3.

We now choose θ_{t_m} to minimise $A(\theta^m, z^m)$ for each θ^{m-1}, z^m. Define

$$B^m(\theta^{m-1}, z^m) = \min_{\theta_{t_m}}\{A(\theta^m, z^m)\}, \tag{3.2}$$

then $B^m(\theta^{m-1}, z^m)$ is the expected loss, located at the right-most decision node for any branch θ^{m-1}, z^m, and is sufficient for all information to the right of it on the tree. So $B^m(\theta^{m-1}, z^m)$ is our expected loss having made decisions θ^{m-1} and having observed z^m, assuming that we intend to make the time t_m decision that minimises expected loss. We can now draw the new rolled back decision tree in figure 3.4 and continue our backwards induction.

Define

$$C^m(\theta^{m-1}, z^{m-1}) = \int_{-\infty}^{\infty} B^m(\theta^{m-1}, z^m) p(z_{t_m}|z^{m-1}, \theta^{m-1}) dz_{t_m}, \tag{3.3}$$

where $p(z_{t_m}|z^{m-1}, \theta^{m-1})$ is our conditional probability distribution for observing z_{t_m}, given decisions θ^{m-1} and observations z^{m-1}. $C^m(\theta^{m-1}, z^{m-1})$ is therefore our

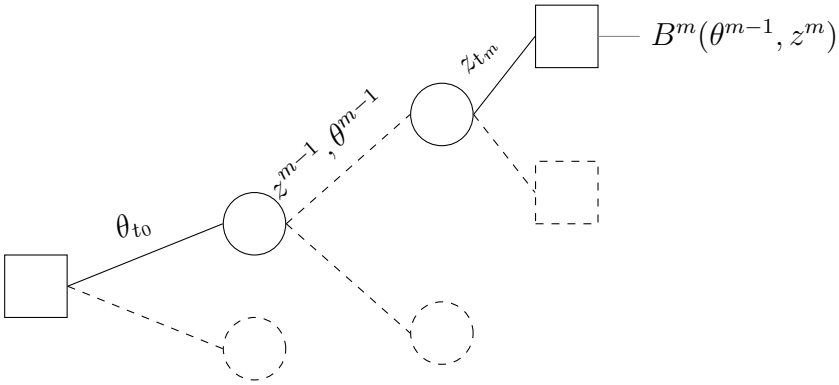

Figure 3.4: This image shows the decision tree from figure 3.3 having been rolled back one step by choosing the θ_{t_m} that minimizes $A(\theta^m, z^m)$ for each θ^{m-1}, z^m.

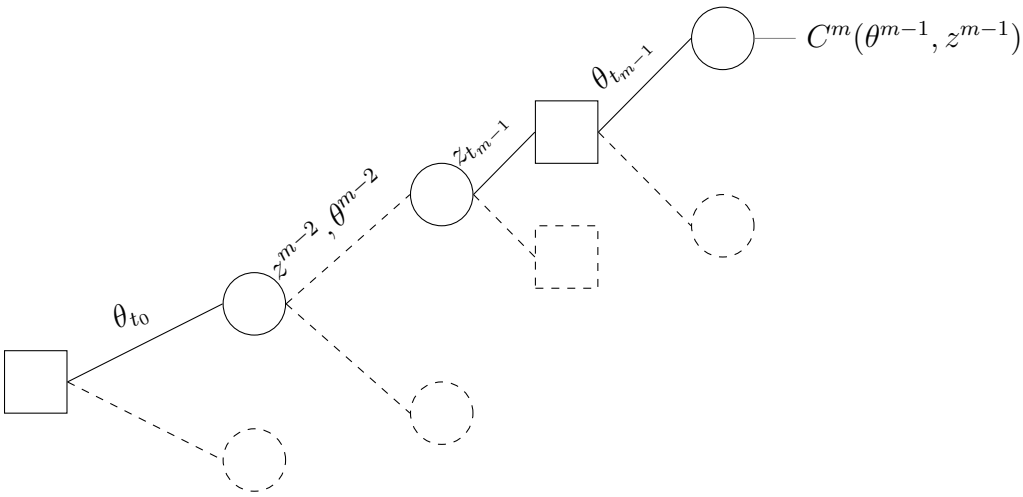

Figure 3.5: This image shows the decision tree from figure 3.4 rolled back a further step, by taking expectations of $B^m(\theta^{m-1}, z^m)$ over the conditional distribution of z_{t_m} given all other decisions and observations, as defined by (3.3).

expected loss having made decisions θ^{m-1} and having observed z^{m-1}. This expected loss is now sufficient for everything to the right of the appropriate chance node on the tree. This is shown on figure 3.5

We continue to compute expectations at each node in this way, working from the right to the left of the tree until we reach the final chance node. For $0 < k < m$ let $B^k(\theta^{k-1}, z^k)$ be our expected loss having made decisions θ^{k-1} and having observed

z^k, and let $C^k(\theta^{k-1}, z^{k-1})$ be our expected loss having made decisions θ^{k-1} and having observed z^{k-1}. Then

$$B^k(\theta^{k-1}, z^k) = \min_{\theta_{t_k}}\{C^{k+1}(\theta^k, z^k)\}, \tag{3.4}$$

with

$$C^k(\theta^{k-1}, z^{k-1}) = \int_{-\infty}^{\infty} B^k(\theta^{k-1}, z^k) p(z_{t_k}|z^{k-1}, \theta^{k-1}) dz_{t_k}, \tag{3.5}$$

and $C^1(\theta_{t_0}, z_{t_0})$ is our expected loss of making today's policy θ_{t_0}, having seen history z_{t_0}. Therefore, $C^1(\theta_{t_0}, z_{t_0})$ is the expected loss surface we wish to be able to visualise.

In order to solve our decision problem we require the probability distributions $p(y|z^m, \theta^m)$ and $p(z_{t_k}|z^{k-1}, \theta^{k-1})$ for $k = 1, \ldots, m$. At the beginning of section 2.3.2 we discuss ways to obtain features of these posteriors using the computer model and a Bayesian emulation technique.

The only method we may use to sample directly from the posterior generated from our beliefs and the data is the full Bayes method. The large-scale MCMC calculation required to produce a forecast distribution is certainly feasible as a one-off. However, as we have stated, it will pose particularly difficult numerical challenges and will take a long time. For any given branch of the decision tree θ^m, z^m we require a large sample from $p(y|z^m, \theta^m)$, meaning a separate MCMC calculation, to be able able to evaluate $A(\theta^m, z^m)$. To evaluate $B^m(\theta^{m-1}, z^m)$ and $C^m(\theta^{m-1}, z^{m-1})$ requires $A(\theta^m, z^m)$ to be computed a large number of times (assuming we obtain results numerically). This means that a prohibitively large number of separate and challenging MCMC calculations are required simply to roll back the tree one time step. This process, if it were ever achievable, must be repeated over m time steps. Hence the full Bayes solution to the policy problem is infeasible, and we cannot obtain $p(y|z^m, \theta^m)$ and $p(z_{t_k}|z^{k-1}, \theta^{k-1})$ for $k = 1, \ldots, m$ using MCMC forecasting methods in general.

3.3.2 Obtaining estimates to the required distributions

We outlined the method of forecasting assuming a Gaussian process emulator and gave detailed calculations and discussion for Bayes Linear forecasting methods. Both approaches enable a mean and variance forecast to be obtained. In the GP case, that

mean and variance can be close to the actual mean and variance of the distribution in which we are interested (assuming we genuinely believe all of the assumptions we have used to generate the conjugate analysis). For example, if we require $p(y|z^m, \theta^m)$, the GP forecast can obtain a mean and variance for $p(y|z^m, \theta^m, \Psi = \hat{\Psi})$. The Bayes Linear adjusted mean and variance in a forecast may be seen as pragmatic estimates for the moments of our required distribution, obtained in a sensible fashion. For example, we may see $E_{z^{m-1}}\left[z_{t_m}; \theta^{m-1}\right]$ and $Var_{z^{m-1}}\left[z_{t_m}; \theta^{m-1}\right]$ (we add the normally suppressed θ^{m-1} to these expressions for clarity) as an approximation to the mean and variance of $p(z_{t_m}|z^{m-1}, \theta^{m-1})$. Alternatively, if a partial prior specification was the only reasonable representation of our beliefs, we might use those adjusted beliefs to help characterise a probability distribution that might be seen as a useful tool to aid decision making, based on a handful of well thought out measures of belief.

In either case, suppose that we use a forecast as an estimate for the moments of our required distribution. How might we then sample from that distribution? The answer is that, without further assumptions, we cannot. However, we may use expert judgements (from experts in the field of the complex system) to elicit features of the distribution that allow our forecasts to characterise it. For example, many distributions are completely characterised by their mean and variance (the Normal and Gamma distributions are two of these). By making such an assumption, we can use our forecasts to generate pragmatic approximations to our required probability distributions.

We saw in section 2.3.4 that, with Bayes Linear methods, we can achieve very quick estimates as our forecasts. Combined with the elicitation mentioned above, we have a way of obtaining an estimate for the required distributions quickly which will prove invaluable for our approach to solving the policy problem.

Justification

We are not limited to distributions completely characterised by their second order moments. If our expert feels that higher order moments are required to fix the distributions or if certain features of the tail of the distribution merit exploration, then we can add these to our forecast. For Bayes Linear methods, we may emulate

many features of the distribution we desire and might theoretically be able to make forecasts that include such features.

Finding the shape of these distributions, assessing how forecast moments characterise them, and even whether the form should be distinct for different θ^m is a difficult elicitation problem. However, we are still asking system experts to comment directly on their specialist field. Thinking about issues such as model discrepancy (a concept which might appear quite abstract) will often be more difficult than thinking about how the system might respond if it has already behaved in a certain way. In the context of building and running the expert model, as well as the costs involved in making policy (and getting it wrong), it is perhaps a problem that is worth the effort.

Lastly, it is important to add that we aim to offer decision support. An integral part of this will be to examine the effects of the distributional assumption on the expected loss surface we present to the policy maker. If the loss surface is sensitive to particular features of the elicitation, these may be re-examined or studied more carefully.

3.3.3 Evaluating expectations

The functions $A(\theta^m, z^m)$ and $C^k(\theta^{k-1}, z^{k-1})$ for $k = 1, \ldots, m$ are expectations with respect to the probability distributions discussed in section 3.3.2. Unless the loss function is an extremely simple expression, as opposed to the series of complex partial differential equations solved numerically that we expect of most computer models (in which case we might be able to compute $A(\theta^m, z^m)$ analytically depending on the form of $p(y|z^m, \theta^m)$), each of these integrals must be evaluated numerically for any θ^m, z^m. Each time we evaluate any of these for a given choice of their inputs, we must compute a forecast (which may take a long time depending on the method). Even with fast methods of forecasting and very efficient numerical integration techniques, we can only evaluate any of the functions $A(\theta^m, z^m)$ or $C^k(\theta^{k-1}, z^{k-1})$ for $k = 1, \ldots, m$ a finite number of times.

The functions $B^k(\theta^{k-1}, z^k)$ for $k = 1, \ldots, m$ pose a further problem. We cannot evaluate the minimum we desire analytically for any k and for any θ^{k-1}, z^k. Esti-

mating the minimum numerically would require a very large number of evaluations of $C^{k+1}(\theta^k, z^k)$ (or $A(\theta^m, z^m)$) at different values of θ_{t_k}.

We cannot solve this decision problem analytically, and numerical estimates of each function require expensive forecasts and numerical integration. For just one θ_{t_0} then, even estimating $C^1(\theta_{t_0}, z_{t_0})$ by brute force seems infeasible. However, a policy must still be made, and the question of how our best expert simulators and data from the real world can guide these policy choices must still be addressed.

As has been described, the functions $A(\theta^m, z^m)$, $B^k(\theta^{k-1}, z^k)$ and $C^k(\theta^{k-1}, z^{k-1})$ for $k = 1, \ldots, m$ are complex expensive functions that we may only evaluate at a limited number of points, but whose behaviour we would like to understand throughout their respective domains. A natural approach to dealing with them would be to treat them as we treat computer models and emulate them. We would then use the emulators in some way to aid the backwards induction of the decision problem. We show in section 3.5 how we may combine emulators for each of the required functions to offer a sensible upper bound on $C^1(\theta_{t_0}, z_{t_0})$. Before describing our algorithm in detail however, we discuss how we might emulate any of our required functions individually.

3.3.4 Combining forecasts

As we are dealing with complex, forecast-dependent expectations, it is difficult to write down a prior emulator for any of the functions in which we are interested. We do, however, have a 'way in' for this problem. For each of the expectations we require, we must solve an integral numerically and compute a forecast. Not only can our numerical integration be either very slow and accurate or fast and approximate, but our forecasting method may be the same. For example, if we have a Bayes Linear second order emulator, we can make really careful, expensive forecasts or use fast forecasts as a cheap approximation.

If we judge that a 'fast' estimate to any of our expectations (using, say, a cheap approximate numerical integration rule and a fast forecast), is informative for an expensive evaluation involving a careful forecast and powerful numerical method, we can use these fast estimates to help us build an emulator. Using fast approximations

to our functions in the way is similar to multi-level emulation, a method from the computer experiment literature. Multi-level methods use cheap, less accurate models to learn about expensive and accurate ones.

Before we present multi-level emulation in detail, we discuss the different speeds of forecast. Bayes Linear forecasting methods are generally quicker than those obtained through integrating out x^* from a Gaussian process emulator and certainly quicker than the full Bayes calculation. If we have decided that the most appropriate description of our computer model requires a Gaussian process or fully Bayesian emulation, we may still value a Bayes Linear version to provide fast and informative estimates of $A(\theta^m, z^m)$ and $C^k(\theta^{k-1}, z^{k-1})$ for $k = 1, \ldots, m$. These estimates can then be tuned using our "heavyweight" probabilistic methods. Whatever our beliefs about the model and the system, then, the Bayes Linear forecasting methods outlined in section 2.3.2 will still form an integral part of our method of decision support.

3.4 Multi-level emulation

3.4.1 Multi-level emulation of computer models

Multi-level emulation, also known as Multifidelity modelling, is a method for building emulators for computer models that exploits faster, cheaper versions of the full simulator. Suppose we wish to emulate an expensive, powerful and 'accurate' simulator denoted $f^a(\zeta)$, where ζ is a vector of inputs of any type. To help build our emulator for $f^a(\zeta)$ we may make many runs on a cheaper 'coarse' version of the simulator that shares much of the physics modelled by $f^a(\zeta)$. We denote this coarse simulator $f^c(\zeta)$. The general idea is to model $f^c(\zeta)$ very well and use that model, combined with a handful of expensive runs of $f^a(\zeta)$, to emulate the accurate model.

The idea of having faster, coarser approximations to a computer model is quite natural. Most computer models (at least the ones we have encountered) consist of a set of mathematical differential equations that must be solved. Whether the solvers use finite element methods or otherwise, they usually involve evaluating the equations at a finite set of points over a user-defined mesh. A coarser model, if one

did not already exist, may often be obtained by 'coarsening' this mesh.

There are different approaches to multi-level emulation, each suitable for the different types of emulator one may wish to build. Leary et al. [61] present two ways of incorporating coarse model runs to model an accurate simulator by kriging. The first is to build an emulator for the difference or the ratio between the accurate and coarse using ordinary kriging methods. The emulator of the differences, denoted $\hat{d}(\zeta)$, interpolates the difference between the models at design points. The authors then write their emulator for the accurate as either

$$\hat{f}^a(\zeta) = \hat{d}(\zeta) + f^c(\zeta)$$

or

$$\hat{f}^a(\zeta) = \hat{d}(\zeta) f^c(\zeta),$$

depending on whether differences or ratios were fitted. Note that the emulator for the accurate model requires a run on the coarse simulator for any evaluation. This may prove problematic if our coarse simulator is not as fast to run as a typical emulator. The second kriging based approach to multi-level emulation discussed by Leary et al. involves using the coarse data as "prior knowledge" and incorporating this knowledge into the kriging model.

The kriging model of the differences (or ratios) between the simulators is established as described in section 2.2.3. A "knowledge layer" is then written

$$\kappa(\zeta) = f^c(W\zeta + w),$$

where W is a diagonal matrix and w is a vector. Using the same set of inputs for both submodels, each element of W and w, along with the usual parameters required to fit the kriging model for the differences $d(\zeta)$, are fitted using maximum likelihood estimation. The emulator is then either

$$\hat{f}^a(\zeta) = \hat{d}(\zeta) + \hat{\kappa}(\zeta)$$

or

$$\hat{f}^a(\zeta) = \hat{d}(\zeta)\hat{\kappa}(\zeta).$$

Bayesian multi-level emulation

The Bayesian approach to multi-level emulation is to write the accurate model as a stochastic process that involves the coarse model and a number of parameters. Given prior judgements about these parameters, runs on both models can be used to update beliefs about the accurate simulator. Kennedy and O'Hagan [57] write their accurate model (assumed to have scalar output) as the sum of the coarse model multiplied by a scalar quantity, ρ, and a stochastic process.

They write

$$f^a(\zeta) = \rho f^c(\zeta) + \varepsilon^a(\zeta) \tag{3.6}$$

where $\varepsilon^a(\zeta)$ is a Gaussian process and is independent of $f^c(\zeta)$. The model is 'trained' using runs from both the accurate and the coarse at the same locations. Kennedy and O'Hagan show how certain assumptions on ρ, $f^c(\zeta)$ and the parameters of $\varepsilon^a(\zeta)$ can lead to a conjugate analysis.

This approach to Bayesian multi-level emulation is extended by Qian and Wu [91] who model their accurate simulator as

$$f^a(\zeta) = \rho(\zeta) f^c(\zeta) + \varepsilon^a(\zeta), \tag{3.7}$$

where, in addition to $\varepsilon^a(\zeta)$, $\rho(\zeta)$ is also a Gaussian process in the inputs ζ. They use MCMC methods to sample a posterior density for the accurate model.

Cumming and Goldstein [22] suggest a multi-level emulation method exploiting the structure of an emulator built for the coarse. Suppose our emulator for the coarse has the standard form of (2.4) and is written

$$f^c(\zeta) = \beta_i g_i(\zeta) + \varepsilon^c(\zeta).$$

Cumming and Goldstein then suggest a form for the accurate emulator that allows us to learn about each of the coefficients for the regression surface of the accurate emulator. Their model is

$$f^a(\zeta) = \rho_i \beta_i g_i(\zeta) + \gamma \varepsilon^c(\zeta) + \varepsilon^a(\zeta) \tag{3.8}$$

(again with $\varepsilon^a(\zeta)$ independent of the coarse), which allows for a greater level of flexibility than, say, the model of Kennedy and O'Hagan in (3.6). Such flexibility may

be appropriate when the different terms in $g_i(\zeta)$ represent features of the physical system that may change differently with the refinement of the model for it. Once the coarse emulator is built, the idea is to use the difference between the coarse and accurate models at a small set of points to learn about the ρ_i's, γ and $\varepsilon^a(\zeta)$.

3.4.2 Emulation of expected losses

Each of the techniques of multi-level emulation we have described is aimed at modelling the output of complex deterministic computer code. An emulator for the complex system model, combined with our beliefs about the loss model and the distributional assumptions we have made, are enough to completely determine any of our required expected losses. However, we can only ever use numerical methods to evaluate these expectations. This means that an evaluation of any of our expectations cannot be treated as a realisation of a deterministic function. We present here a method of multi-level emulation for integrals that we must evaluate numerically.

Suppose we wish to emulate a scalar-valued function

$$\pi(\zeta) = \int \eta(\zeta, \tau) d\tau, \tag{3.9}$$

where $\eta(\zeta, \tau)$ is such that we must evaluate the integral numerically and τ is a vector of arbitrary length. The function $\eta(\zeta, \tau)$ itself may be difficult or expensive to evaluate and may require estimation. For example, consider $A(\theta^m, z^m)$ defined in (3.1), the first integral required for solving the policy problem. Here ζ is (θ^m, z^m), τ is y and $\eta(\cdot, \cdot)$ is $L(y, \theta^m)p(y|z^m, \theta^m)$.

Suppose we have a 'coarse' method of estimating $\pi(\zeta)$. For example, we may use Monte Carlo methods with a handful of samples. We may also use an approximate evaluation of $\eta(\zeta, \tau)$. When solving the expected loss integrals, for example, we may use a Bayes Linear fast forecast to characterise $p(y|z^m, \theta^m)$. We denote this 'coarse' function $\pi^c(\zeta)$. At the same time, we have a careful or 'accurate' method of estimating the integral, denoted $\pi^a(\zeta)$, such as using Quadrature or perhaps a very large Monte Carlo sample. We may also use a careful method of evaluating $\eta(\zeta, \tau)$: for example, when applying this to $A(\theta^m, z^m)$ we may use a careful Bayes Linear forecast or even a fully probabilistic method to characterise $p(y|z^m, \theta^m)$.

Both versions approximate $\pi(\zeta)$ with some numerical error that we wish to 'smooth out'. Final inference about $\pi(\zeta)$ will be obtained through inference about $\pi^a(\zeta)$. The approach is to build a good emulator for the coarse and link only the interesting parts to the accurate in our multi-level emulation. We suppose that $\pi^c(\zeta)$ contains a signal informative for the value of $\pi(\zeta)$ and a mean zero 'numerical error' $\delta^c(\zeta)$, and we judge, for simplicity, that

$$\pi^a(\zeta) = \pi(\zeta) + \delta^a(\zeta), \tag{3.10}$$

where $\delta^a(\zeta)$ represents pure numerical error from the accurate integration, with $E\left[\delta^a(\zeta)\right] = 0$ and $Corr\left[\delta^a(\zeta), \delta^a(\zeta')\right] = 0$ for $\zeta \neq \zeta'$. We note that this relationship is different to that given in, for example, (3.6), because $\delta^a(\zeta)$ will be part of our model for $\pi^a(\zeta)$ and we intend to obtain expectations of $\pi(\zeta)$ by smoothing the uncorrelated error in our accurate emulator. We view $\pi^a(\zeta)$ as an observation of $\pi(\zeta)$ made with error. Note that this judgement depends on our choice of method for evaluating $\eta(\zeta, \tau)$. For example, if we chose a Bayes Linear 'careful' forecast to characterize $p(y|z^m, \theta^m)$ in our accurate evaluations of (3.1), our emulator for $A(\theta^m, z^m)$ would depend on this approach to forecasting and, more subtly, on the design we used to build the emulator for the complex system model. We discuss this fully in section 3.7.

3.4.3 Bayes Linear calculations for multi-level emulation

We present a Bayes Linear approach to this multi-level emulation. The approach is similar to that of Cumming and Goldstein [22], although we simplify the calculations by only allowing the regression surface from the coarse emulator to vary by a multiple ρ on the accurate. We do this only to simplify the calculations; if the problem required it, this assumption could be relaxed.

Suppose our coarse emulator is

$$\pi^c(\zeta) = \beta_j g_j(\zeta) + \varepsilon^c(\zeta) + \delta^c(\zeta), \tag{3.11}$$

where $g(\zeta)$ is a known vector of basis functions in ζ, β is a vector of coefficients to be obtained through linear fitting, $\varepsilon^c(\zeta)$ is a correlated residual process representing

behaviour not captured by the global regression and $\delta^c(\zeta)$ is uncorrelated error. This uncorrelated error will be related to the accuracy of our numerical integration technique, but may also be partly comprised of a residual in any 'non active' elements of ζ. If our integration is over many dimensions, we may decide to use the method of active variables (see section 2.2.6 and equation (2.16)) to reduce the dimensions over which we must integrate. In this case $\delta^c(\zeta)$ would represent our uncertainty as to the effect of the non-active elements of ζ as well as numerical integration error. We write the accurate model as

$$\pi^a(\zeta) = \rho\beta_j g_j(\zeta) + \gamma\varepsilon^c(\zeta) + \varepsilon^a(\zeta) + \delta^a(\zeta), \tag{3.12}$$

where ρ and γ are scalar multipliers, $\varepsilon^a(\zeta)$ is a weakly stationary process capturing effects only present in the accurate model, and $\delta^a(\zeta)$ is the uncorrelated error from the accurate calculation. The variance of $\delta^a(\zeta)$ should be smaller the variance of $\delta^c(\zeta)$ as we are using a better numerical integration technique.

We assume that we may obtain enough coarse evaluations to choose $g(\zeta)$ and to determine the values of β. We therefore treat these as known quantities once fitted. By observing the residuals from this regression and by estimating the variance of $\delta^c(\zeta)$ through replicate sampling or otherwise, we may write down correlation lengths and a variance for $\varepsilon^c(\zeta)$ (we could use variogram fitting (see Cressie [20]) to obtain these if preferred). We assume

$$Cov\left[\delta^c(\zeta), \delta^c(\zeta')\right] = Cov\left[\delta^a(\zeta), \delta^a(\zeta')\right] = 0. \tag{3.13}$$

We take an n-point design and compute $\pi^c(\zeta_{[i]})$ and $\pi^a(\zeta_{[i]})$ for $i = 1, \ldots, n$, where $\zeta_{[i]}$ denotes the value of ζ at the ith design point. Let

$$E_i^c = \varepsilon^c(\zeta_{[i]}), \qquad i = 1, \ldots, n.$$

If we have E_i^c, then, at design points

$$\pi^a(\zeta_{[i]}) = \rho\beta_j g_j(\zeta_{[i]}) + \gamma E_i^c + \varepsilon^a(\zeta_{[i]}) + \delta^a(\zeta_{[i]})$$

is a linear model with coefficients ρ, γ and residual $\varepsilon^a(\zeta_{[i]}) + \delta^a(\zeta_{[i]})$. Given prior judgements on ρ and γ as well as variance and correlation parameters for the residual, we can adjust $\pi^a(\zeta)$ by the difference between accurate and coarse runs.

At design points $\zeta_{[i]}$ for $i = 1, \ldots, n$, assuming we have each of the E_i^c,

$$E_D\left[\pi^a(\zeta_{[i]})\right] = E_D\left[\rho\right]\beta_j g_j(\zeta_{[i]}) + E_D\left[\gamma\right]E_i^c + E_D\left[\varepsilon^a(\zeta_{[i]})\right] + E_D\left[\delta^a(\zeta_{[i]})\right], \quad (3.14)$$

where

$$\begin{aligned}
D_j &= \pi^a(\zeta_{[j]}) - \pi^c(\zeta_{[j]}) \\
&= (\rho - 1)\beta_i g_i(\zeta_{[j]}) + (\gamma - 1)E_j^c + \varepsilon^a(\zeta_{[j]}) + \delta^a(\zeta_{[j]}), \\
E\left[D_j\right] &= (E\left[\rho\right] - 1)\beta_i g_i(\zeta_{[j]}) + (E\left[\gamma\right] - 1)E_j^c + E\left[\varepsilon^a(\zeta_{[j]})\right] + E\left[\delta^a(\zeta_{[j]})\right], \\
Var\left[D_j\right] &= Var\left[\rho\right](\beta_i g_i(\zeta_{[j]}))^2 + Var\left[\gamma\right](E_j^c)^2 + Var\left[\varepsilon^a(\zeta_{[j]})\right] \\
&\quad + Var\left[\delta^a(\zeta_{[j]})\right] + 2Cov\left[\rho, \gamma\right](\beta_i g_i(\zeta_{[j]}))E_j^c.
\end{aligned}$$

Let $H = Var\left[D\right]^{-1}(D - E\left[D\right])$ and $G_{ij} = g_i(\zeta_{[j]})$ then

$$E_D\left[\rho\right] = E\left[\rho\right] + Var\left[\rho\right](\beta_i G_{ij})H_j + Cov\left[\rho, \gamma\right]E_j^c H_j,$$

and similarly

$$E_D\left[\gamma\right] = E\left[\gamma\right] + Cov\left[\gamma, \rho\right](\beta_i G_{ij})H_j + Var\left[\gamma\right]E_j^c H_j.$$

We adjust ρ and γ in this way using the design points, and then we may calculate the expectation of $\pi^a(\zeta)$ away from design points via

$$E_D\left[\pi^a(\zeta)\right] = E_D\left[\rho\right]\beta_j g_j(\zeta) + E_D\left[\gamma\right]E\left[\varepsilon^c(\zeta)\right] + E_D\left[\varepsilon^a(\zeta)\right], \quad (3.15)$$

where $E\left[\varepsilon^c(\zeta)\right]$ is obtained having adjusted beliefs about the coarse correlated residuals by all observed coarse runs. Note that there is no $E_D\left[\delta^a(\zeta)\right]$ term in (3.15) because we specified that $\delta^a(\zeta)$ have mean zero and be uncorrelated in ζ. Through assumption (3.13) we can write the adjusted variance away from the design points as

$$\begin{aligned}
Var_D\left[\pi^a(\zeta)\right] &= (\beta_i g_i(\zeta))^2 Var_D\left[\rho\right] + \left[Var\left[\gamma\right] + E\left[\gamma\right]^2\right]Var\left[\varepsilon^a(\zeta)\right] \\
&\quad + E\left[\varepsilon^c(\zeta)\right]^2 Var_D\left[\gamma\right] + 2(\beta_i g_i(\zeta))E\left[\varepsilon^c(\zeta)\right]Cov_D\left[\rho, \gamma\right] \\
&\quad + Var_D\left[\varepsilon^a(\zeta)\right] + Var\left[\delta^a(\zeta)\right].
\end{aligned}$$

From (3.10) our adjusted expectation for $\pi(\zeta)$ is

$$E_D\left[\pi(\zeta)\right] = E_D\left[\pi^a(\zeta)\right] - E_D\left[\delta^a(\zeta)\right]$$

at design points and $E_D\left[\pi^a(\zeta)\right]$ everywhere else.

Obtaining the coarse correlated residuals at design points

Let $R_j^c = \pi^c(\zeta_{[j]}) - \beta_k g_k(\zeta_{[j]})$ be the residuals from the coarse runs, then the function

$$E_{R^c}\left[\varepsilon^c(\zeta) + \delta^c(\zeta)\right] = E_{R^c}\left[\varepsilon^c(\zeta)\right] + E_{R^c}\left[\delta^c(\zeta)\right]$$

interpolates the coarse residuals. Therefore the adjusted expectation of the correlated residual surface $E_{R^c}\left[\varepsilon^c(\zeta)\right]$, whilst not interpolating the coarse correlated residuals exactly at design points, is the natural choice of surrogate. We compute $E_{R^c}\left[E^c\right]$ and assume $E^c \approx E_{R^c}\left[E^c\right]$. Note that the strength of this relationship is measured by the size of $Var_{R^c}\left[E^c\right]$. If $Var_{R^c}\left[E^c\right]$ is quite small, we can be quite confident in using $E_{R^c}\left[E^c\right]$ as a surrogate for E^c. Due to the nature of multi-level fitting $Var_{R^c}\left[E^c\right]$ will almost always be very small because the number of points making up R^c, can be very large. In practice, we have found stating $E^c = E_{R^c}\left[E^c\right]$ to be a weaker assumption than stating $Var\left[\beta\right] = 0$, which we have already done.

3.5 Sequential Emulation

In this section we propose a method for emulating an upper bound on $C^1(\theta_{t_0}, z_{t_0})$, our expected loss of making policy today. Our approach is to emulate expectations at each of the chance nodes in sequence using the multi-level methods we described in section 3.4.2. Emulators are used to choose an intervention strategy at each t_i and expected losses are re-emulated under that strategy.

The algorithm we propose uses emulators to remove each of the m decision nodes representing the policy interventions one by one, from right to left on the decision tree. We present the algorithm in full detail in section 3.6. Before writing the algorithm out in detail, we discuss the rationale and detail the key steps, as well as offering an overview of the methodology.

3.5.1 Strategy at t_m

Our first objective in a Sequential Emulation of a decision tree is to choose an intervention strategy for the mth intervention point. This strategy is a function of θ^{m-1} and z^m so that, whatever decisions we have made up until that point and

whatever we have observed of the system, we can compute the optimal strategy going forward. We label the time t_m intervention strategy $\lambda_{t_m}(\theta^{m-1}, z^m)$. Ideally we require

$$\lambda_{t_m}(\theta^{m-1}, z^m) = arg \min_{\theta_{t_m}} A(\theta^m, z^m),$$

or written another way

$$B^m(\theta^{m-1}, z^m) = A((\theta^{m-1}, \lambda_{t_m}(\theta^{m-1}, z^m)), z^m).$$

As it is not possible to compute $A(\theta^m, z^m)$ exactly (and even our good approximations take a long time), we cannot ensure that this condition is satisfied. Our method is to emulate $A(\theta^m, z^m)$ as a function of θ^m, z^m and use the emulator to locate the minimum. In essence, this approach is optimization with emulators which was discussed in section 2.2.3.

Using the multi-level methods we described in section 3.4.2 we construct an emulator for $A(\theta^m, z^m)$ where our two versions of the calculation use different numerical integration techniques, as well as fast and careful forecasting, to provide a coarse and an accurate estimation. Using our accurate emulator we may compute $E\left[A(\theta^m, z^m)\right]$ relatively quickly. This expected loss, calculated directly from our emulator, may be treated as a loss because we specified that $L(y, \theta^m)$ was in utility units.

Having emulated $A(\theta^m, z^m)$ then, we define

$$\lambda_{t_m}(\theta^{m-1}, z^m) = \arg \min_{\theta_{t_m}} \{E\left[A(\theta^m, z^m)\right]\}. \tag{3.16}$$

Note that $A((\theta^{m-1}, \lambda_{t_m}), z^m)$ is an upper bound on $B^m(\theta^{m-1}, z^m)$. We write this upper bound

$$B^m_{\lambda^m}(\theta^{m-1}, z^m) = A((\theta^{m-1}, \lambda_{t_m}), z^m). \tag{3.17}$$

Our t_m strategy is now viewed as fixed for all θ^{m-1}, z^m by λ_{t_m} so that we may remove the decision node at t_m from our decision tree. We show this in figure 3.6. We now emulate our upper bound for $B^m(\theta^{m-1}, z^m)$, namely $B^m_{\lambda^m}(\theta^{m-1}, z^m)$, by repeating the process used to emulate $A(\theta^m, z^m)$, fixing $\theta_{t_m} = \lambda_{t_m}$. This is an expected loss conditioned on λ_{t_m}. We could simply use our first emulator for $A(\theta^m, z^m)$, evaluated at $\theta_{t_m} = \lambda_{t_m}$, as an emulator for the required upper bound. However, we feel that the quality of our emulator for the upper bound is better

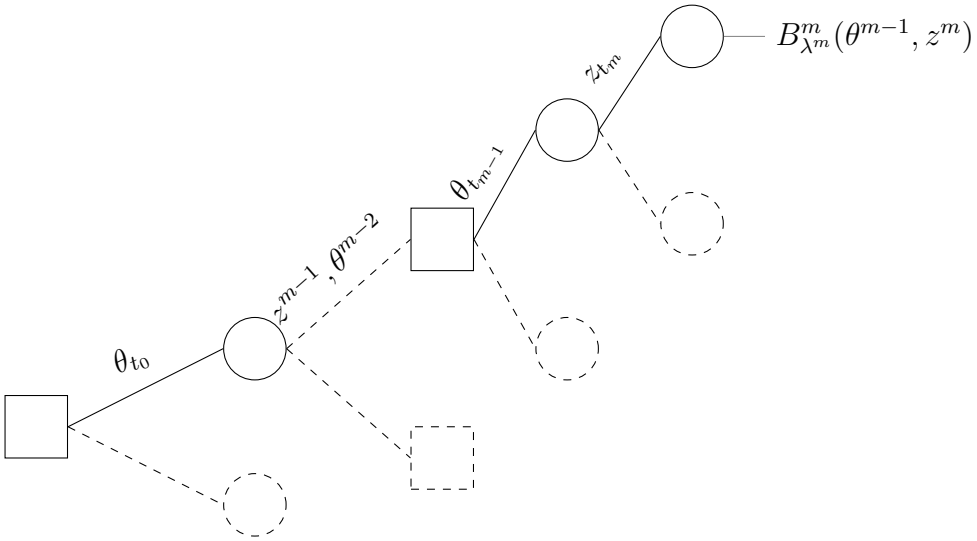

Figure 3.6: This image shows our decision tree after we have fixed $\lambda_{t_m}(\theta^{m-1}, z^m)$. The right-most node is now replaced by the upper bound $B^m_{\lambda^m}(\theta^{m-1}, z^m)$ as defined in (3.17).

served by evaluating expected losses at $\theta_{t_m} = \lambda_{t_m}$ and re-emulating to capture any local behaviour around our chosen strategy as carefully as possible. The loss $B^m_{\lambda^m}(\theta^{m-1}, z^m)$ is in utility units so we may treat $E\left[B^m_{\lambda^m}(\theta^{m-1}, z^m)\right]$, as calculated from our new emulator for $B^m_{\lambda^m}(\theta^{m-1}, z^m)$, as an expected loss.

3.5.2 Removing the next decision node

In order to remove the decision node at t_{m-1}, we require $C^m(\theta^{m-1}, z^{m-1})$, the expected loss at the chance node adjacent to it on the right. The approach is to use the emulator for $B^m_{\lambda^m}(\theta^{m-1}, z^m)$ to emulate an upper bound on $C^m(\theta^{m-1}, z^{m-1})$. Once we have this emulator, we define a strategy as before by using the emulator to locate it.

Define the operator $\tilde{E}\left[\Gamma\right]$ to mean an expectation obtained by emulating the function Γ and computing the expectation of the emulator. We define $C^m_{\lambda^m}(\theta^{m-1}, z^{m-1})$ as

$$C^m_{\lambda^m}(\theta^{m-1}, z^{m-1}) = \int \tilde{E}\left[B^m_{\lambda^m}(\theta^{m-1}, z^m)\right] p(z_{t_m}|z^{m-1}, \theta^{m-1}) dz_{t_m}, \tag{3.18}$$

which is an expected loss because L is measured in utility units. $C^m_{\lambda^m}(\theta^{m-1}, z^{m-1})$

is an emulator for an upper bound on our expected loss for having made decisions θ^{m-1} and having observed z^{m-1}, conditioned on time t_m strategy $\lambda_{t_m}(\theta^{m-1}, z^m)$.

Our goal is now to choose $\lambda_{t_{m-1}}(\theta^{m-2}, z^{m-1})$ to select the $\theta_{t_{m-1}}$ that minimizes $C^m_{\lambda^m}(\theta^{m-1}, z^{m-1})$. Even if the form of our emulator for $B^m_{\lambda^m}(\theta^{m-1}, z^m)$ and our distributional assumption for $p(z_{t_m}|\theta^{m-1}, z^{m-1})$ are such that for any θ^{m-1} and z^{m-1} we could compute $C^m_{\lambda^m}(\theta^{m-1}, z^{m-1})$ analytically, the requirement of a forecast in order to characterize (or sample from) $p(z_{t_m}|\theta^{m-1}, z^{m-1})$ still renders $C^m_{\lambda^m}(\theta^{m-1}, z^{m-1})$ a very expensive function. We therefore emulate this and define

$$\lambda_{t_{m-1}}(\theta^{m-2}, z^{m-1}) = \arg\min_{\theta_{t_{m-1}}} \{\tilde{E}\left[C^m_{\lambda^m}(\theta^{m-1}, z^{m-1})\right]\}.$$

By fixing strategy at time t_{m-1} via $\lambda_{t_{m-1}}$ we remove another decision node from the decision tree.

3.5.3 Determining time t_k intervention strategy

Suppose we have determined strategies

$$\lambda^{k+1} = \lambda_{t_{k+1}}, \ldots, \lambda_{t_m},$$

so that our decision tree is as shown in figure 3.7 and we define

$$B^m_{\lambda^{k+1}}(\theta^k, z^m) = \int L(y, (\theta^k, \lambda^{k+1}))p(y|z^m, \theta^k, \lambda^{k+1})dy. \tag{3.19}$$

Define

$$C^m_{\lambda^{k+1}}(\theta^k, z^{m-1}) = \int \tilde{E}\left[B^m_{\lambda^{k+1}}(\theta^k, z^m)\right] p(z_{t_m}|z^{m-1}, \theta^k, \lambda^{k+1})dz_{t_m} \tag{3.20}$$

and

$$C^j_{\lambda^{k+1}}(\theta^k, z^{j-1}) = \int \tilde{E}\left[C^{j+1}_{\lambda^{k+1}}(\theta^k, z^j)\right] p(z_{t_j}|z^{j-1}, \theta^k, \lambda^{k+1})dz_{t_j}, \tag{3.21}$$

for $j = k+1, \ldots, m-1$. We then define

$$\lambda_{t_k}(\theta^{k-1}, z^k) = \arg\min_{\theta_{t_k}}\{\tilde{E}\left[C^{k+1}_{\lambda^{k+1}}(\theta^k, z^k)\right]\}. \tag{3.22}$$

Any function we compute involving an emulator is technically also an emulator, so that simply evaluating the function could qualify as emulation. When referring to this methodology, we reserve the term emulator to mean either the computer model

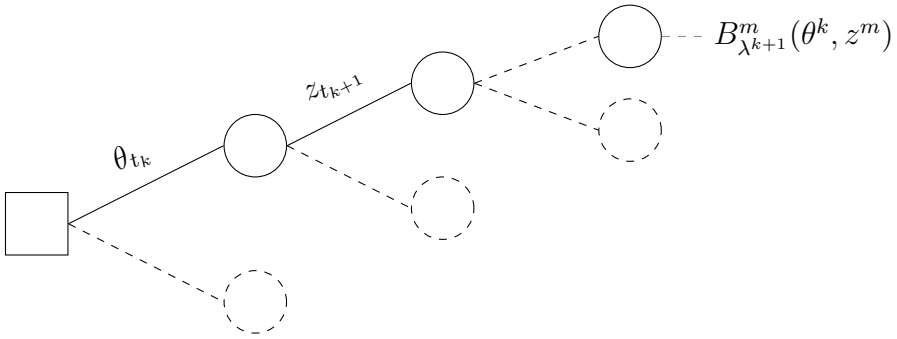

Figure 3.7: This image shows a sub tree of our decision tree, starting from the kth decision, after we have fixed our first $m-k$ intervention strategies, thus removing the corresponding decision nodes. The dashed line connecting the two right-most chance nodes represents observations $z_{t_{k+2}}, \ldots, z_{t_m}$

emulator or an emulator built using multi-level methods on an expectation integral. Objects derived from these emulators, such as expectation functions, we assume to be easily computable once the main emulator is built and so when counting the number of emulators required, these objects are not included. Note, then, that given λ^{k+1} we require $(m-k+1)$ emulators to obtain $\lambda_{t_k}(\theta^{k-1}, z^k)$.

3.5.4 The upper bound on our expected loss surface

Our aim is to provide an upper bound on $C^1(\theta_{t_0}, z_{t_0})$. Supposing we have used the above methods to determine all of our intervention strategies; $\lambda^1 = \lambda_{t_1}, \ldots, \lambda_{t_m}$. Then, in constructing $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$ defined as in (3.21), we have an emulator for an upper bound for this expected loss surface. Having established our first intervention strategy, $\lambda_{t_1}(\theta_{t_0}, z_{t_1})$, we must emulate $B^m_{\lambda^1}(\theta_{t_0}, z^m)$, $C^m_{\lambda^1}(\theta_{t_0}, z^{m-1})$,...,$C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$, meaning that $(m+1)$ emulators are to be constructed using multi-level methods. We require, in total, $\frac{1}{2}(m+1)(m+2)$ multi-level emulations to provide the desired emulator for our upper bound on our expected loss surface.

3.6 The Sequential Emulation Algorithm

We now present a detailed version of the Sequential Emulation algorithm. We begin by outlining the preliminary steps required before using our methods to address the decision tree. We will then state the sequential emulation algorithm.

3.6.1 Preliminaries

We begin with a computer simulator for the complex system, $f(x, \theta)$ and a loss function $L(y, \theta)$. We assume that our methods of emulation will be Bayesian and that we have already addressed the design question, obtained runs F, and exhausted our ability to run the computer model.

P1

Build an emulator for $f(x, \theta)$. The emulator should adequately reflect our beliefs about the model and should be as accurate as we can make it, particularly around plausible values of x^*. We require a second order form for our emulator so that we may exploit the speed of Bayes Linear forecasting later. This does not restrict us in our choice of emulator for the computer model; it simply means that if a second order emulator is not an adequate reflection of all of our beliefs, we are able to derive one from our full (or enlarged) specification. For Gaussian process methods this should be straight forward. A fully probabilistic emulator may not have a simple, easy to evaluate second order form because there will be distributions on the correlation parameters. A second order emulator may be constructed using a fully probabilistic emulator by, for example, using expectations (and variances where appropriate) of each of the required parameters, gained by examining the updated marginal distributions.

If the problem and our beliefs have warranted construction of a GP or fully probabilistic emulator, this will be the one used directly for learning about our expected losses and optimal intervention strategies. The second order form will be required only to provide relatively fast forecasts that are informative for our true beliefs about the system.

P2

Specify or elicit beliefs about x^*, $\eta(\theta)$ and e. If our specification is fully probabilistic, these beliefs must be linked to the priors for our emulator before it is updated by the model runs. If we intend only to use Bayes Linear methods for forecasting, then we require a prior probability distribution for x^* and only a second order specification for the remaining quantities.

P3

Select both 'accurate' and 'fast' forecasting methods. When performing the Sequential Emulation algorithm we will require $\frac{1}{2}(m+1)(m+2)$ multi-level emulations. Each of these is an expectation with respect to a probability distribution for future states or observations of the complex system. We will be using Bayes Linear fast forecasts (see section 2.3.4) for all coarse versions of the expectations in our emulations. This will allow us to obtain estimates more cheaply and quickly than if using any other forecasting method. Which method of fast forecasting we use should be decided here. Our careful forecasting method will depend on our emulator and the amount of computing power we have available. It must also be specified in advance of performing the Sequential Emulation.

P4

Decide how forecasts will be used to sample from the required distributions. If our careful forecasting method is a fully Bayesian MCMC calculation, this sampling is part of the forecast. Otherwise, as discussed in section 3.3.2, we have either only a mean and a variance estimate, or a few chosen moments and quantiles. For our fast forecasts and coarse approximations, at least, we must use the ideas from section 3.3.2 to decide how the forecasts will characterize each distribution.

P5

Specify numerical integration methods. We must decide on 'coarse' and 'accurate' numerical integration methods for each expectation in our Sequential

Emulation. Although each expectation is univariate, the integral may be many-dimensional, leaving Monte Carlo or MCMC as the only realistic choices.

3.6.2 Sequential Emulation

Having completed the above preliminary steps we are ready to begin our Sequential Emulation of the decision tree.

S1

Construct a multi-level emulator for $A(\theta^m, z^m)$. To construct this emulator we require a large sample of coarse runs (the term 'coarse' is given meaning through P3, P4, P5). We must design these runs and use linear fitting, residual diagnostics, our own judgements and perhaps other methods to construct (3.11). We must then obtain a number of coarse and accurate evaluations at the same locations. We then use these, along with some of our own judgements, to construct the accurate emulator through (3.12), (3.14) and (3.15). We may use emulator diagnostics to test the performance of our fit and obtain further runs to 'fine tune' if neccessary.

We provide a detailed example of this kind of Bayes Linear multi-level emulation within a Sequential Emulation setting in Chapter 4.

S2

Define $\lambda_{t_m}(\theta^{m-1}, z^m)$ **as in (3.16)**.

S3

Construct a multi-level emulator for $B^m_{\lambda^m}(\theta^{m-1}, z^m)$. This follows the same process as for **S1**, except we set

$$\theta^m = (\theta^{m-1}, \lambda_{t_m}(\theta^{m-1}, z^m)).$$

We are therefore emulating a function of θ^{m-1} and z^m where the first step for each evaluation of either coarse or accurate version is to compute $\lambda_{t_m}(\theta^{m-1}, z^m)$.

S4

Construct a multi-level emulator for $C^m_{\lambda^m}(\theta^{m-1}, z^{m-1})$. We are evaluating

$$\int \tilde{E}\left[B^m_{\lambda^m}(\theta^{m-1}, z^m)\right] p(z_{t_m}|z^{m-1}, \theta^{m-1}) dz_{t_m}$$

and obtain coarse and accurate estimates to the required conditional distribution through the decisions made at **P3** and **P4**. The multi-level emulation procedure is the same as with **S1** and **S3**, but requires us to emulate a function of fewer parameters.

S5

If $m = 1$ **proceed to *S11*. Else set** $k = m - 1$ **and go to *S6***

S6

Define $\lambda_{t_k}(\theta^{k-1}, z^k)$ **using the emulator for** $C^{k+1}_{\lambda^{k+1}}(\theta^k, z^k)$ **and (3.22).**

S7

Construct a multi-level emulator for $B^m_{\lambda^k}(\theta^{k-1}, z^m)$ **as defined in (3.19).**

S8

Construct a multi-level emulator for $C^m_{\lambda^k}(\theta^{k-1}, z^{m-1})$ **as defined in (3.20). Set** $j = m - 1$

S9

Construct a multi-level emulator for $C^j_{\lambda^k}(\theta^{k-1}, z^{j-1})$. **Let** $j = j - 1$

S10

If $j \geq k$ **go back to S9. Else if** $k = 1$ **proceed to *S11*. Else set** $k = k - 1$ **and go back to *S6*.**

S11

Stop

The last emulator constructed for $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$ is an emulator for our upper bound on the expected loss surface for policies made today.

3.7 Policy Support

Once obtained, we may use $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ to provide decision support for policy makers. $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ is a function of our policy today that is relatively cheap to evaluate, hence we may plot it. The ability to visualize an upper bound for the policy maker's expected loss surface is very useful. The policy maker may observe which parameters of θ_{t_0} have most effect on the expected loss, as well as seeing which areas of the decision space drive most of the variability.

If our resources are spent or our time has run out completely, we may use our upper bound to locate our 'optimal' policy for today by choosing θ_{t_0} to minimize $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$. This choice of policy (note this is λ_{t_0} as defined in (3.22)) is our estimate of the true minimum of $C^1(\theta_{t_0}, z_{t_0})$. It takes future observations and policy interventions into consideration, as well as giving a serious treatment to the discrepancy between the system and our computer model for it. Whilst using our upper bound to locate optimal policy directly may seem to be the most obvious and natural thing to do, if we have sufficient time/resources we can use the surface and the methodology to provide considerably more. We describe some of our methods of policy support using Sequential Emulation here.

3.7.1 Pruning

We begin with a decision space containing $(m+1)$ decisions, $\theta_{t_0}, \ldots, \theta_{t_m}$, each potentially being vectors of multiple parameters. The emulators for the computer model and for $A(\theta^m, z^m)$ have been built using designs over the entire space as we seek to provide a good global fit for any feasible policy. Each of the subsequent emulators built during our Sequential Emulation of the decision tree must also fit their 'target functions' well over the whole range of feasible decisions.

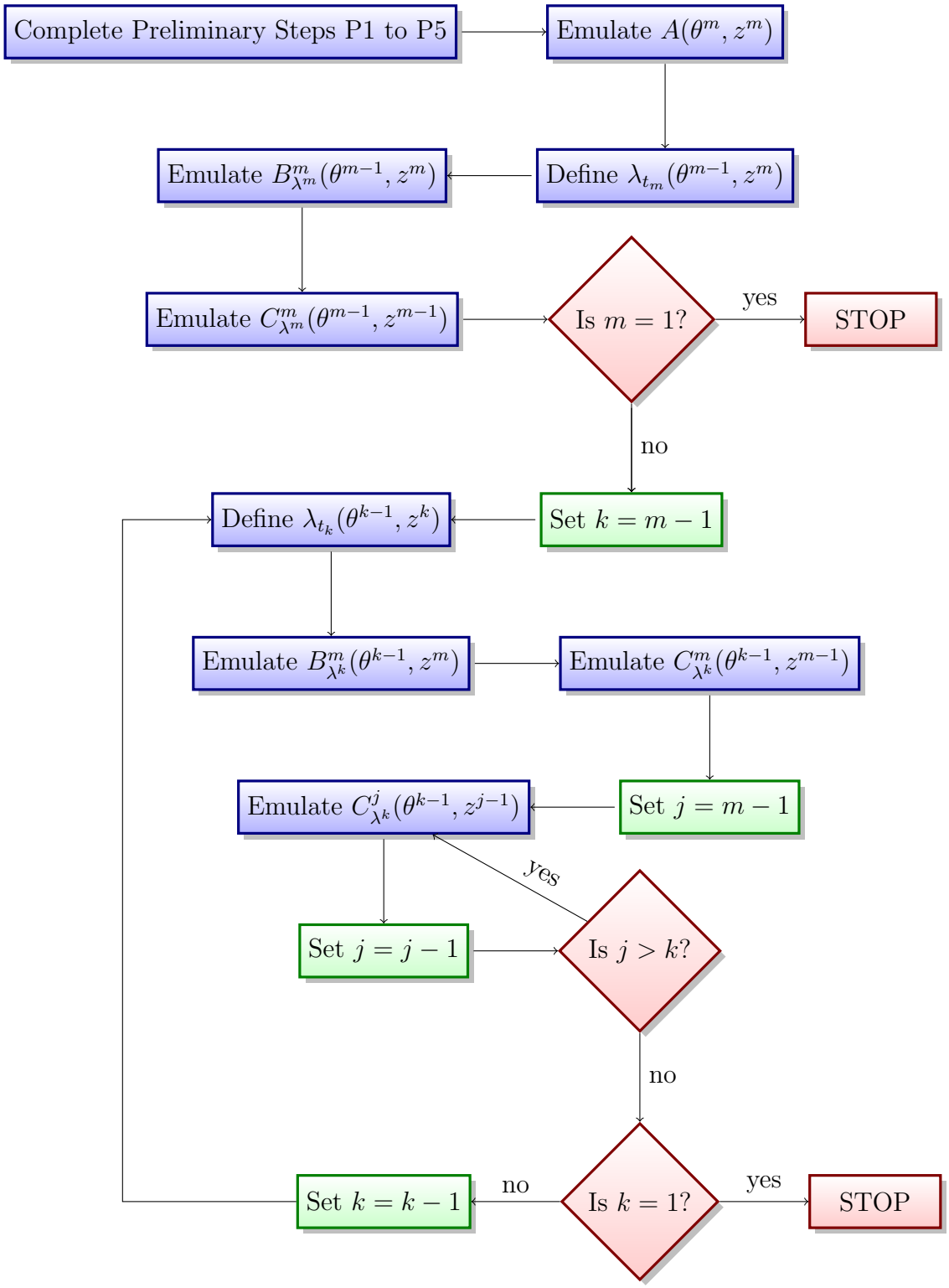

Figure 3.8: A flow chart describing the steps of the Sequential Emulation algorithm. The labels P1-P5 refer to the preliminary steps of the algorithm. The labels S1-S10 refer to the main steps of the algorithm.

Inevitably, the larger the input space of a function we wish to emulate, the more difficult it is to build an emulator that performs well globally. Ideally, we would like to reduce the size of the decision space to provide more accurate emulators for the types of policy we are prepared to consider. This idea is similar to History Matching (described in section 2.3.1) for computer models. We use History Matching in computer model emulation to reduce the size of the input space and refocus our emulators to improve our analysis.

Refocusing is an important idea in many applications that involve the analysis of computer models. For example, some of the Galaxy simulation studies performed at Durham University (see, for example, Vernon and Goldstein [108]) involve an iterative search for settings of the model inputs that make the model behave consistently with the real world via History Matching. Having removed areas of the input space that are deemed implausible matches to real-world observations of Galaxy formation, the emulators for GALFORM are refocussed on the reduced input space. This means that the model is run once again over a new design on the reduced input space and the emulators are rebuilt. The current analysis of GALFORM has been refocused four times.

Another important use of refocusing techniques is in the optimization literature, where the goal is to look for a setting of the control inputs to a simulator that optimizes the output. Optimization algorithms will often use emulators to refocus the search for the minimum of the simulator around the minimum of the emulator, whilst also sampling the function in areas of high uncertainty to insure against returning a local minimum.

Using emulators created during a Sequential Emulation we may reduce the size of the decision space and refocus our analysis to improve our emulators via *pruning*. Pruning involves looking at plots for our final expected loss surface and at what we call *strategy plots*, to locate areas of the decision space that lead to intolerably high expected losses or that are not present in any of our strategies. Our time k strategy is $\lambda_{t_k}(\theta^{k-1}, z^k)$. A strategy plot is a plot of λ_{t_k} against any of its inputs or against expected loss for fixed values of its inputs. By looking at strategy plots we may notice that there are entire areas in the domain of θ_{t_k} that are never visited by λ_k.

We may also notice that for a range of key decisions we are interested in, there are areas of the domain of θ_{t_k} in which we find it highly implausible that the minimum expected loss could be located. As these areas are not involved in our Sequential Emulation, we may consider removing them from the domain of θ_{t_k} altogether.

In looking at our final expected loss surface we may observe expected losses that are intolerably high for certain values of θ_{t_0}. If their performance in relation to other potential policies is particularly poor (or if their loss is above a certain tolerance) we may consider removing these areas of the domain of θ_{t_0} from our feasible set of decisions. We may also consider elements of the risk profile (see section 3.7.3) in areas of the decision space that are candidates for removal. If the risk profile for a given policy is 'less favourable' than that for a set of θ_{t_0} with smaller expected losses, we may be happy to remove that policy (and others similar to it) from the tree.

Each time we reduce the domain for any given policy or intervention we are, in effect, 'pruning' the decision tree. By beginning at θ_{t_0}, pruning the decision tree, then moving through the tree and pruning the remaining subtrees using strategy plots, we may dramatically reduce the size of the decision space.

The surface $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ and each of the intervention strategies offer a way of showing policy makers what their beliefs and the functions they have given us imply about the expected loss of their decisions and the way they should act in the future. Which areas of the decision space to remove will be a decision that should rest with the policy maker. They may set tolerances or criteria that make pruning automatic, or they may choose to combine some of the other methods of support offered, such as risk profiling or sensitivity analysis, to choose suitable areas for pruning.

As was noted in section 3.4.2, the results of our Sequential Emulation depend implicitly on the design (Ω, Θ) used to select runs on our computer model. The more our design tells us about $f(x^*, \theta)$ for the θ in which we are interested, the better our policy support will be. If our Sequential Emulation was not the last word on the policy problem, we may use a pruned tree to refocus our analysis. Ideally we may still make runs on our computer model $f(x, \theta)$. If this is the case then we may make further runs on the reduced space to refocus our emulator for the computer

model. The refocussed emulator would be more accurate for the decisions we still consider feasible and so our forecasts will be more accurate. Simply facilitating a reduction of the size of the decision space and a refocussing of the computer model emulator provides a convincing argument for performing a Sequential Emulation of the decision tree and certainly qualifies as policy support. Even just one pruning exercise may be enough to refocus wider studies of the system, the economic impacts and the social welfare functions within the policy problem.

If our policy support is required to go further, we may now perform a Sequential Emulation on the pruned tree (using the refocussed computer model emulator if available). By evaluating each of the expectations on the tree over a reduced domain, our Sequential Emulation will be more accurate and could offer further insight into the impacts of the policies within that domain. The Sequential Emulation may also be be faster depending on how much of the space has been pruned.

If we intend to prune and refocus many times to offer policy support, it may be wise to use more approximate methods in each of our multi-level emulations for the first few Sequential Emulations. If our beliefs demand fully Bayesian MCMC sampling when forecasting, for example, it might be prudent not to use those methods until we have pruned the tree first. We may use Bayes Linear methods, even in our careful forecasts, while pruning and save the very expensive calculations for the pruned tree.

Related to pruning is dimension reduction, which may be possible using the Sequential Emulation. If, for some strategy λ_{t_k}, the strategy plots indicated that optimal strategy was (approximately) invariant over previous decisions and observations, we may consider removing that intervention point altogether and taking a default action at time t_k. The observations made between the last intervention point and the one removed, z_{t_k}, would be absorbed into $z_{t_{k+1}}$. It may be that fewer interventions than we have attempted to include are appropriate because a meaningful intervention can only be made if 'enough' observations have been made to inform us of any system trends. For example, consider the climate CO_2 decision problem discussed at the beginning of this chapter and suppose we planned to make an intervention every year based on observed temperatures. Climate temperature

may be increasing as part of some emissions-driven trend, or as the result of natural variability in the climate system. After only one year's observations, it would be unreasonable to expect to have observed any trend at all, hence an intervention would be inappropriate. Not only would such dimension reduction make emulation of the computer model and Sequential Emulation easier, but it represents very useful policy support. If it can be demonstrated that a particular intervention is not required, this will save significant amounts of time and money for policy makers.

We have been deliberately vague regarding exactly how one chooses which parts of the decision tree to prune (or indeed when it is acceptable to consider an intervention point redundant). If every emulation of the loss function was an actual expected loss, then perhaps a given criterion here would be meaningful. However, as discussed previously, the loss function may be only one plausible parametrization of a loss model. We must also investigate the impact of using other plausible parametrizations, as well as investigating the sensitivity of our methods to our belief specification. Any decision taken regarding pruning the decision tree must also take these issues into account.

3.7.2 Sensitivity and scenario analysis

Throughout the analysis of the decision problem, potentially aided by experts, we make a number of statistical judgements. For example, a variance for the model discrepancy is a judgement or statement of belief regarding how informative the model is for the system. Decision makers themselves (or economic modellers) have also made judgements when constructing their social welfare function. These judgements often come in the form of parametrizations and expressions designed to capture their true preferences over the set of feasible consequences of their decisions. It is important, if possible, to offer advice to policy maker regarding which of these judgements may have significant effects on the expected loss surface, particularly around its minimum.

Sequential Emulation offers a way of obtaining a visualization of the loss surface for different values of key judgements. By observing which quantities alter the loss surface when changed, the decision maker will be able to focus his resources on

providing the best possible expert judgement for those quantities. Furthermore, if the surface remains relatively unchanged despite extreme values of a given quantity, he can focus his energy elsewhere. These ideas also apply to strategy plots, whose reaction to different judgements may not only indicate which judgements are important, but also which intervention points are the most crucial in determining today's expected loss for a given policy.

The methodology also allows the decision maker to view a number of consequences of his choice or parametrization of the loss function. For complicated problems, this loss function will often be an economic model and will therefore be subject to similar kinds of uncertainty as our model for the system. A fully probabilistic sensitivity analysis for policy problems with intervention is unrealistic (see Oakley [81]). However, observing $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ for parametrizations of the loss function could be very useful. Indeed, where scientists or policy makers differ in their beliefs about the reaction of the economy to (say) climate change, this visual aid may point to judgements that hardly affect the loss surface and to those key judgements where more agreement is needed.

Scenario analysis involves setting the parameters of the loss function to mimic a given scenario (such as a population explosion; see Kann and Weyant [54] for more details). The methodology we have described seems particularly suited to scenario analysis, in that a picture of the loss surface for each considered scenario can be compared, and any decisions that appear robust under many different scenarios can be considered more carefully.

As mentioned previously, robust decisions are often particularly important to policy makers when their preferences are not certain and when scientists disagree about the economic impact of future scenarios. Finding robust decisions, or even discovering that there are no decisions robust to all plausible beliefs regarding the interaction of the system with the economy and our preferences, is a key part of decision support. Any criteria for pruning may be subject to certain conditions on robustness, or it may be that Sequential Emulation is used solely to locate the key judgements that require further investment, study and agreement. If this is the nature of the required policy support, it may be foolish to consider any methods of

emulation and of forecasting other than the Bayes Linear methods. This is because a purely Bayes Linear Sequential Emulation is far faster than one that uses alternative (albeit more powerful if our beliefs are genuinely probabilistic in nature) methods of forecasting. A sensitivity (or scenario) analysis such as the one described would require many separate Sequential Emulations, which would likely be infeasible if our forecasting methods were slow. We believe that the use of a (perhaps more pragmatic and approximate) Bayes Linear Sequential Emulation is appropriate when searching for sensitive parameters and robust policies, as opposed to choosing optimal policy directly, which should use all of our prior knowledge. Note that if we are unprepared or unable to give a fully probabilistic specification of our beliefs about the simulator and the discrepancy, but we are able to specify second order beliefs, the a Bayes Linear Sequential Emulation is appropriate in any case.

3.7.3 Risk profiling

Having performed a Sequential Emulation, we have fixed a strategy at each of the m intervention points (written λ^1). For a given policy, θ_{t_0}, we may construct the risk profile, conditioned on λ^1, by forward sampling. The *risk profile* is the distribution of $L(y, (\theta_{t_0}, \lambda^1))|z_{t_0}, \theta_{t_0}, \lambda^1$, and we sample from it by using each of the distributional assumptions used in our Sequential Emulation.

We first sample from $p(z_{t_1}|z_{t_0}, \theta_{t_0})$ and call this sample $\hat{z}_{t_1}$. Using our set of strategies we compute

$$\hat{\lambda}_{t_1} = \lambda_{t_1}(\theta_{t_0}, (z_{t_0}, \hat{z}_{t_1})).$$

We then obtain a $\hat{z}_{t_2}$ by sampling from $p(z_{t_2}|\hat{z}_{t_1}, z_{t_0}, \theta_{t_0}, \hat{\lambda}_{t_1})$ and use it to compute $\hat{\lambda}_{t_2}$. Continuing in this manner and sampling each of the m observations and computing the m strategies in turn, we finally sample a $\hat{y}$ from $p(y|\hat{z}^m, \theta_{t_0}, \hat{\lambda}^1)$ and compute $L(\hat{y}, (\theta_{t_0}, \hat{\lambda}^1))$.

This process gives one draw from the distribution from which we are sampling. We repeat the process many times to obtain a risk profile for a given policy. As this is an expensive calculation, we may only perform it for a handful of chosen policies. Which policies merit further investigation may be decided by observing the expected loss surface. In addition, the policy maker may have a number of "default" decisions

for which he wishes to obtain such a risk profile.

The expectation of our risk profile can be used as a diagnostic tool to see how well our Sequential Emulation is doing. For a given policy, if we are doing well, the expectation of its risk profile should lie near the surface $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$. If it is far away for given decisions, we may wish to re-emulate around those decisions. We discuss and illustrate this idea further in section 4.7.2.

3.8 Discussion

3.8.1 Feasibility of Sequential Emulation

Sequential Emulation is designed to be a practical means to providing policy support. We are required, however, to construct $\frac{1}{2}(m+1)(m+2)$ emulators in addition to building an emulator for the computer model. Each of these emulations becomes more challenging as m and the dimension of both observations and decisions increases. Feasibility of the methodology for large problems will (as with all methodologies concerning computer experiments) depend on computing power and resources. We consider here how the number of interventions, the dimension of each decision and the dimension of y_{t_i} for any i, affect feasibility. We may also consider how the length of the sequence of observations of y before each intervention affects the calculations, although this may be thought of as increasing the dimension of each y_{t_i}.

A large number of intervention points means that the Sequential Emulation algorithm requires many more individual emulations. We consider this problem more carefully later and restrict ourselves here to considering how increased dimensionality affects each individual emulation. Suppose we have m intervention points, and the decision at intervention point i is θ_i, an $m_\theta-$vector of values. We suppose, for the sake of this illustrative argument, that each decision is a vector of the same length, m_θ. Then the vector θ of all decisions to be taken is a vector of $(m+1)m_\theta$ quantities. Let m_x be the number of model inputs to the simulator. Therefore our emulator for the computer model is an emulator over $m_x + (m+1)m_\theta$ variables. Further, we suppose for argument's sake, that each system value at intervention point i is a

vector with m_y quantities, then our emulator is a projection from $m_x + (m+1)m_\theta$ dimensional space into $(m+2)m_y$ dimensional space.

Building the computer model emulator is our first difficulty in tackling a problem with large m_y, m or m_θ, although it is not our primary concern as the Sequential Emulation methodology assumes an emulator for the computer model can, and will, be built. Whilst a number of techniques, such as careful choice of active variables, may be used to reduce the effective size of m_x, we must emulate the simulator as a function of all of our decisions if we wish to provide policy support. Note that if a particular decision is not 'active', in that it has little or no effect on any of the model's outputs, we would question whether or not an intervention at that point was necessary at all. The curse of dimensionality, then, means that m, m_θ and m_y must be sufficiently small to enable us to obtain enough runs on our complex simulator in order to build a useful emulator for it. This problem is common to all computer experiment methodology.

Suppose then that m, m_θ and m_y are small enough so that we can build an informative emulator for $f(x, \theta)$. We now consider other impacts that the size of m, m_θ and m_y may have on our methodology. Sequential Emulation requires us to be able to emulate $\frac{1}{2}(m+1)(m+2)$ complex integrals that are functions of subsets of system observations and decisions. The function we must emulate with the fewest number of inputs is a function of m_θ variables. This is our final expected loss surface for making a decision today given that all m downstream intervention strategies are fixed. The function with the largest number of inputs is that representing the right-most node on the decision tree and is a function of $m(m_y + m_\theta) + m_\theta$ variables.

To perform such an emulation would require a number of evaluations of our expected loss much greater than $m(m_y + m_\theta) + m_\theta$, and this loss itself is an integral over $m_y(m+2)$ dimensions. This task may seem arduous for large m_y, m, m_θ, but we must first consider the context. We have already emulated a very complex function (our computer model), projecting $m_x + (m+1)m_\theta$ dimensions onto $(m+2)m_y$ dimensions, and we also expect to be able to integrate this emulator numerically over m_x dimensions. Furthermore, we can assume that there will have been a significant amount of resources and man hours directed at building the computer model in the

first place. If the model was built in order to assist policy makers in important decision problems, then we should not baulk at spending a large amount of time and resources addressing the actual policy problem.

If it is indeed possible to emulate our computer model effectively and to integrate out the best input, then we should be able to emulate a one-dimensional expected loss as a function of many variables if we can obtain reasonable estimates for the desired integrals in a reasonable amount of time. The concept of what is deemed a 'reasonable' amount of time must depend on the total time and resources available to us and the importance of the problem. The best methodologies for numerical integration break down quickly as the dimension of the problem increases. Quadrature, for example, is exceptionally accurate for low dimension problems, but requires too many points in high dimensional space to be feasible. However, Monte Carlo methods are always available and their accuracy (measured by the variance of the estimator, at least) is independent of dimension. Using less accurate methods of numerical integration may be unavoidable for many real-world problems anyway, and as long as we are able to account for any error properly in our emulation, the size of $(2+m)m_y$ or m_y (the dimension of the integrals) is not our main cause for concern. Our principle issue lies in performing the integration enough times to be able to fit a meaningful regression on a maximum of $m(m_y + m_\theta) + m_\theta$ variables.

For large problems, it is unlikely that we will be able to obtain 'sufficiently informative' Monte Carlo estimates fast enough. There might, however, be sufficient structure in our emulators to allow parts of the integration to be done analytically. We may consider emulating the loss function and integrating each of the regression terms independently. A regression surface in any of the emulators we might build is likely to be a function of monomials and other small subcollections of the variables we are integrating out. Therefore, we might break the coarse integration down into a series of much faster integrations and save any large and difficult integrations (such as that for the residual process) for the handful of accurate evaluations we must make. Whilst this could further 'coarsen' our coarse emulators, our multi-level emulation would still reflect our uncertainties as long as we carefully considered how the coarse and accurate evaluations related to each other. How one computes these expectation

integrals, when the integrands themselves are functions of emulators with respect to certain distributions, is a difficult and open research question, whose answer will be heavily problem dependent.

When taken in the context of emulating our computer simulator (which could take days or weeks to run just once), these types of calculation should be possible. To speed up the process, we may make our 'fast' forecasting very quick using the separability assumptions that allow us to provide decision-dependent forecasts across the decision space, having integrated out x^* just once. Separability, though a strong assumption, may be a worthwhile tool for fast forecasting, even if we do not judge the assumption appropriate for the careful calculation. As m_θ, m, m_y increase, each emulation becomes more difficult and, because more runs are required for building emulators, the process becomes slower.

We have assumed m, m_θ and m_y are small enough so that we are able to build a meaningful emulator for $f(x, \theta)$. We have argued that, if this is the case, we should be able to build our Sequential Emulators through particular choices of fast forecasting methods and/or breaking down each integrand into integrable, lower dimensional components. The time this takes per emulator will only be increased due to the number of runs required. In a serious analysis, parallel computing should be able to help relieve this concern.

A remaining concern is; how will the size of m, m_θ and m_y affect accurate calculations? The answer to this depends on the method used to produce accurate calculations in each of our multi-level emulations. It is likely that full Bayes methods quickly become infeasible as the dimensions increase, and if we are performing careful Bayes Linear calculations, the computation time will depend on whether or not we can assume separability (in full or in part) and what degree of numerical accuracy is deemed appropriate. It is worth noting that although each multi-level emulation is problem-dependent, once a particular method is working well for these loss integrals, it should work well for the rest of the Sequential Emulation. This is because the functions all rely on forecasts of similar quantities evolving in time. It is also likely that a certain degree of automation may be applied to the problem.

3.8.2 Treatment of the loss model

Our Sequential Emulation treats the loss model as a function by fixing its input parameters x_L at some value. In fact, the loss model $L(x_L, y, \theta)$ has much in common with our computer model $f(x, \theta)$ and may even be expensive to run. Aside from the decision and system outcome parameters, θ and y, the loss model contains two kinds of inputs; parametrized by x_L. The first subcollection of inputs within x_L define the preferences of the policy makers and populations by using a social welfare function for economic output and any other quantifiable damage to the real world. Sensitivity and scenario analysis should provide adequate treatment of these preference parameters when policy makers do not fully understand their preferences. The second subcollection drives the behaviour of the economy and the growth of populations as well as describing how outcomes of the complex system 'damage' the economy.

This second group has a lot in common with the model parameters of our computer simulator. They represent the parameters for an attempt by the modellers to describe a complex system (the economy) and its interaction with the system modelled by $f(x, \theta)$. In effect, the loss model is then two submodels. We have a model that uses physics and economics to describe the state of the economy under different policies and outcomes, and a social welfare function whose input is the state of the economy (and maybe some aspects of y) and whose output is a utility. We have not handled our uncertainty for this first submodel appropriately in our analysis. Further extension of our methodology and the Sequential Emulation algorithm should consider how well this submodel represents the economy.

One idea is to disaggregate the two submodels and to treat the economic part as another computer model. The best input approach could be applied and we may properly handle our uncertainties regarding this model through careful consideration of a model discrepancy and through emulation. We might even have access to real world data regarding economic growth under different outcomes from the complex system. For example, we are continually gathering data regarding how global warming is affecting the planet and the monetary costs involved in damages and abatement. This data may be used to adjust our beliefs about the real economy.

Under this approach, an expectation for future values of the economy under different decisions and outcomes may be calculated, and this expectation may then be passed into our social welfare function to give an expected loss. Careful treatment of the loss model is an interesting area of further work to be explored. We believe such a treatment is possible and that it would fit into our Sequential Emulation framework, enabling more powerful policy support.

Chapter 4

Sequential Emulation for a simplified climate policy problem

In this chapter we apply Sequential Emulation to a simplified version of a real world climate policy problem. Using an intermediate complexity climate model and a loss function derived from an Integrated Assessment model, we illustrate how Sequential Emulation may be used to provide policy support for a CO_2 emissions problem. The example allows the expected loss of different CO_2 abatement strategies to be explored under the assumption that at one fixed point in the future, global temperature may be observed and policy may be revised.

In giving this example, we aim to illustrate how our Sequential Emulation methods work in practice, as well as giving insight into the types of questions that must be answered and the types of problems that must be overcome in order to perform such an analysis. We also give examples of the types of policy support that the results of the analysis may provide.

It should be noted that, whilst the material in this chapter provides a useful illustration of our approach, it does not constitute a serious attempt at providing support for a real world climate policy problem. Such support would require a careful emulation of a powerful climate model, a large amount of climate data on many different variables, significant input from climate and economic experts as well as state of the art economic models and social welfare functions.

In section 4.1 we describe the policy problem under consideration. Section 4.2

describes the computer models that we will use to learn about climate and its impact on the economy. In section 4.3 we build an emulator for the climate model we are using. Section 4.4 describes our treatment of the economic loss model we have. In sections 4.5 and 4.6 we present the preliminary and main steps of our Sequential Emulation of the decision tree defined by our example. In section 4.7 we give examples of some of the policy support ideas introduced in section 3.7, as applied to the example.

4.1 The CO_2 emissions problem

Our illustration concerns the topical issue of how to abate CO_2 emissions optimally in order to manage the effects of global warming properly, whilst taking into account abatement costs and our utility for emissions-related consumption. We show how our methods may be used to provide decision support for this type of policy problem given a simple climate model, climate data, a model for the economy (that incorporates the economic impact of climate change), and a social welfare function.

We consider only one climate variable; the global mean surface air temperature anomaly. This is defined as the temperature increase as measured from the beginning of the 20th Century. In order to be compatible with our available data and our computer models, we consider policy to be made as if the year were 1995. Our example then, is a demonstration of decision support for a policy maker in 1995 and not a policy maker today.

We imagine that the policy maker wishes to set a global CO_2 emissions strategy in 1995 and call this θ_{t_0}. In 40 years the policy maker plans to observe the surface air temperature anomaly and to set a new emissions strategy θ_{t_1}. He can then no longer intervene with the climate system and the population must live with the consequences of future temperatures. To simplify our calculations, we only consider temperature anomalies at 1995, 2035 and 2095, labelling these y_{t_0}, y_{t_1} and y_{t_2} respectively. The last of these, y_{t_2}, is taken as representative of future temperatures while y_{t_0} represents the historical temperature anomaly, whose observation is z_{t_0}. If we were providing policy support for a real-world policy maker we would use a large

series of temperatures to represent both historical and future values. For example, y_{t_2} may be a vector of annual temperatures from 2036 until some distant future date, whose economic state we no longer hold any preferences over. A heavyweight analysis may also include many more intervention points, with each observation consisting of temperature over a number of years.

The CO_2 emission strategies we must choose are controlled via two decision parameters; θ_{t_0} and θ_{t_1}. The first, θ_{t_0}, specifies a ten yearly reduction to 'business as usual' (BAU) emissions from 1995 in the form of a damping coefficient. This can be thought of as one minus a reduction rate; so, for example, $\theta_{t_0} = 0.7$ corresponds to a ten yearly emissions cut of 30% from BAU. This determines an emissions curve from 1995 to 2095. The second parameter, θ_{t_1}, indicates a further ten yearly reduction to this curve beginning in 2035. This reduction is parametrized in the same way as θ_{t_0}. The BAU curve represents global expected emissions increases (if left unchecked) and was fitted using yearly emissions data from 1950 and the A1F1 SRES future emissions scenario freely available from [48]. Our decisions are parametrized in this way in order to be compatible with our models; introduced in section 4.2. Again, in an important analysis, this parametrization would be approached differently to ensure that uncertainty in the subjective BAU scenario was properly handled so that an emissions policy would lead to a distribution of potential real-world emissions.

Our climate data, z_{t_0}, along with the observation of y_{t_1} that we intend to make in 2035, is a single temperature anomaly. Our observed z_{t_0} is 0.45^o and is taken from the second report from the Intergovernmental Panel on Climate Change (IPCC) [46]. This observation was taken in 1994, but for convenience we treat this as observed in 1995. We set the variance of the mean zero observation error, e_{t_0}, so that an observation more than 0.22^o away from the true increase is a 3 standard deviation event. This is in approximate agreement with the third IPPC report [45], whose improved methods of data processing still give an anomaly for the year 2000 quoted as "$0.6^o \pm 0.2$". We make the variance of e_{t_1} the same.

Our goal is to use Sequential Emulation with the computer models we introduce in section 4.2 to provide decision support for the problem of choosing θ_{t_0}. To ensure our example clearly illustrates the practical application of our methodology, we shall

follow the algorithm of section 3.6 precisely. Prior to this, we introduce the computer models.

4.2 The Computer Models

A Sequential Emulation for this policy problem requires a climate model, $f(x, \theta)$, and a loss function, $L(y, \theta)$, modelling the economic reaction to CO_2 abatement as well as the damage caused by different future temperatures. The last function must include a social welfare function for economic states that outputs a utility.

Our climate model is C-GOLDSTEIN; described by its authors as an "efficient 3-D ocean-climate model" [29]. For our loss function we use the computer model DICE-99. The models DICE and C-GOLDSTEIN have been combined before as part of a Community Integrated Assessment Model called GOLDICE (see Drouet et al. [28]). Although Sequential Emulation uses both models to provide decision support, our methods leave the models completely separate so that only real-world temperature judgements are ever inputted into the loss function. We now introduce these models in further detail.

4.2.1 C-GOLDSTEIN

C-GOLDSTEIN is an efficient, intermediate complexity climate model with highly simplified physics, but with 3-D ocean dynamics. A detailed description of the encapsulated physics and the numerical solvers used can be found in Edwards and Marsh [29]. We treat C-GOLDSTEIN as a 'black box' climate computer model that is informative for the real climate. It has 27 model input parameters and requires a decadal CO_2 emissions series from 1995 to 2095 as a decision input. We calculate this emissions series using θ_{t_0}, θ_{t_1} and our BAU curve so that we treat θ_{t_0} and θ_{t_1} as decision inputs to the model.

For one choice of the model inputs, C-GOLDSTEIN is run to equilibrium for a model time of 4000 years. This process is called 'spin up' and is common to many climate models. Historical global CO_2 concentrations from the year 1850, combined with our future emissions scenario, are then added to the model as it is run to 2100.

C-GOLDSTEIN takes around an hour to run on a desktop computer, although most of that time is required for spin up. Once the model is spun up, it can be run into the future for many different emissions scenarios at little extra cost (under two minutes per run).

C-GOLDSTEIN's outputs include the yearly rate of the meridionnal overturning current in Sv and the yearly global mean air temperature. In order to have model data corresponding to the climate variables we are studying, we focus only on air temperature. We compute the differences between output at 1995, 2035 and 2095, from output at 1900 for any run to give a model temperature anomaly at each time point. These are labelled $f_{t_0}(x)$, $f_{t_1}(x, \theta_{t_0})$ and $f_{t_2}(x, \theta_{t_0}, \theta_{t_1})$ respectively.

To simplify emulation of the computer model, we treat it as a function with only 3 model inputs and fix the remaining 24 at default settings. In a case study, or even an example designed to illustrate computer model emulation, we would not make such a simplification. We may consider only 3 variables active, but would handle uncertainty in the other 24 in a way such as that discussed in section 2.2.6. The illustration of our methods is focussed on Sequential Emulation following an emulation of the computer model. Therefore, simplifying the computer model and our treatment of it prevents us becoming bogged down in what is a very hard problem in itself. The inputs we choose to vary are Climate Sensitivity (x_1), Ocean Vertical Diffusivity (x_2) and Atmospheric Moisture Diffusivity (x_3). These were suggested to us by our model experts as being influential for global mean temperature. We do not concern ourselves with any physical meaning of these variables and treat C-GOLDSTEIN as a 'black box' function of inputs x_1, x_2, x_3, θ_{t_0} and θ_{t_1}.

4.2.2 DICE

DICE is an integrated assessment model by Nordhaus and Boyer [80] that has been used in a number of the integrated assessment examples cited in section 3.2. A given CO_2 emissions strategy is inputted to the model and DICE uses this to compute a global mean surface air temperature anomaly for each decade-sized time step using a simple climate model. Each calculated temperature is used to compute a 'damage value', which is used along with cost functions for CO_2 abatement and

a Douglas-Cobbs formula, to compute the output of the world economy for each decade. Output per capita is converted to consumption, which is then passed to a social welfare function in order to give a utility for a particular policy.

The version of DICE we use is DICE-99 (as opposed to the more modern and arguably better DICE-07 [78]) in order to be compatible with our version of C-GOLDSTEIN. For DICE-99 the climate damage is

$$D(t) = d_1 T(t) + d_2 T(t)^2,$$

where t indicates the time step, $T(t)$ is the temperature at time step t and d_1 and d_2 are model parameters. The output of the world economy at time t is

$$Q(t) = \Omega(t)(1 - b_1(t)\mu(t)^{b_2})A(t)K(t)^{\gamma}P(t)^{1-\gamma},$$

where

$$\Omega(t) = \frac{1}{1 + D(t)}.$$

$A(t)$, $K(t)$ and $P(t)$ represent productivity, capital stock and population respectively, each governed by their own submodels. $\mu(t)$ is the time t emission control rate given as 1 minus the ratio of emitted CO_2 to BAU emissions at a given time step. The function $b_1(t)$ represents an evolving cost on abatement and has its own submodel.

Global consumption $C(t)$ is

$$C(t) = Q(t) - I(t)$$

where $I(t)$ represents investment. Utility for a particular strategy is given as

$$W = \sum_t P(t) \log\left(\frac{C(t)}{P(t)}\right) R(t),$$

where

$$R(t) = \Pi_{v=0}^{t}(1 + \rho(v))^{-10}$$

is a rate of discounting, controlled by the parameters of $\rho(v)$ and designed to reflect our preferences for high consumption today, as opposed to in the future.

As DICE is an integrated assessment model, $T(t)$ is calculated at each time step from the emissions strategy using a simple climate model. We remove this aspect of

DICE to allow real-world temperatures and our judgements regarding them to drive our decision support. We require a temperature at each decade and for the years $1995 - 2095$ we linearly interpolate between the three input temperature values, y_{t_0}, y_{t_1}, and y_{t_2}, that are part of our analysis. Our temperature series will therefore cease in 2095, yet DICE can be run much further into the future until the temporal discounting rate, R(t), damps any utility contribution to zero. To account for our utility for leaving the climate in state y_{t_2} we calculate our utility, U_{t_2} for the 2095 model economy and write

$$\sum_t P(t) \log \left(\frac{C(t)}{P(t)} \right) R(t) + \sum_{n=0}^{\infty} \frac{U_{t_2}}{(1+r)^n}. \tag{4.1}$$

The quantity r is a rate parameter representing a discounting for our utility for the wealth of future generations. We felt this was an intuitive, flexible and pragmatic way of accounting for the wealth of future generations. We can ensure the second term in (4.1) decays quickly by choosing large r to reflect a social preference for consumption today to the detriment of future generations. We can ensure the opposite social preference by choosing small r and allowing the second term to dominate (4.1).

A full list of all of the equations for quantities we have not given can be found in Appendix B of Nordhaus and Boyer [80]. Our decoupled version of DICE-99, coded in R, is given in full in Appendix C. Although DICE is a loss model as opposed to a function of only y and θ, we fix each of its parameters at default values in order to make it compatible with the Sequential Emulation methods present in chapter 3. This is discussed further in section 4.5.1.

4.3 Emulation of C-GOLDSTEIN

We begin our handling of the preliminaries to the Sequential Emulation algorithm at **P1** and describe construction of an emulator for the computer model C-GOLDSTEIN.

4.3.1 Model runs and the form of the emulator

Our treatment of C-GOLDSTEIN will be a Bayes Linear one and we therefore require a second order emulator of the form

$$f_i(x, \theta) = \beta_{ij} g_j(x, \theta) + u_i(x, \theta), \tag{4.2}$$

$i = 1, \ldots, 3$. In order to construct such an emulator, we first require an appropriate vector of regressors $g(x, \theta)$. We then require $E[\beta]$ and $Var[\beta]$, as well as a set of parameters defining the residual process $u(x, \theta)$, designed to encapsulate our beliefs about the model.

To help us make these judgements and to construct the required objects we had access to a very limited bank of runs. A Latin Hypercube of size 48 in the model input variables and a 1050 Hypercube in the decision variables (split into 38 blocks of 25 and 10 blocks of 10) was used to obtain 1050 runs of the model based on 48 different spin ups. We break our design down in this way in order to exploit the relatively fast times involved in running the model for a given abatement strategy once it has been spun up. This was discussed in section 4.2. These runs were completed by Gemma Stephenson at the University of Southampton and were to be our only source of information for model emulation. As a substitute for the expert judgements we would hope to obtain in a case study, we use the 38 spin ups with 25 runs each to build a prior emulator for the model. We then adjust this emulator using the remaining 10. We choose to use the majority of our data to construct a prior that we then adjust by a smaller amount of data, in order to allow an illustration of the basic forecasting methodology described in chapter 2. Whether or not our emulator would be more accurate or fit for purpose if we had used all of the runs to construct the emulator without performing any adjustment, or if we had made an 'ignorant' prior specification and adjusted by everything, is not a question we address. The implications of this choice for our final emulator are discussed in section 4.3.5.

4.3.2 Choosing the regressors

Our first task is to choose the basis functions $g(x, \theta)$ using our 38 spin ups. We achieve this by using a combination of graphical judgements and ordinary least-squares fitting. We attempt to fit independent linear models to each output and to use the collection of monomial terms fitted across all outputs as our vector $g(x, \theta)$. These linear models will not have the same interpretability as the standard regression model because the residuals cannot be assumed to be independent draws from the same Normal distribution. Indeed, the residuals will be part of $u(x, \theta)$ which we shall define as a correlated residual process in x and θ. We use ordinary least squares here only as a tool for model selection and to obtain priors for quantities that will be updated.

We begin by fitting a model to $f_1(x)$. Figure 4.1 shows a scatter plot matrix of the output data F_1 against each of the three model inputs. By eye there seem to be linear effects in each of the 3 variables. As we only have 38 data points, we felt it inappropriate to fit any higher order terms without risk of over fitting. Our goal is to choose a regression surface that fits well to the majority of our output and to use the correlated residual process to capture any effects that remain. We therefore choose

$$f_1(x) = \beta_{11} + \beta_{12}x_1 + \beta_{13}x_2 + \beta_{14}x_3 + u_1(x), \tag{4.3}$$

but base our fits on only 37 of the points. We remove the influential point, seen in the panels for x_2 against F_1 and x_3 against F_1 in figure (4.1), from the data used to train this model so that our regression line captures the linear trends evident in the plot as well as possible. We do not ignore this point entirely, but use it later to make a judgement regarding the magnitude of the variance on our residual process.

Figure 4.2 shows the data corresponding to 950 evaluations of $f_2(x, \theta_{t_0})$ on 38 distinct x values for different θ_{t_0}. From these plots we see quadratic effects in θ_{t_0} along with evidence of the same linear relationship between the model output and the x variables. We therefore decide to fit

$$f_2(x, \theta_{t_0}) = \beta_{20} + \beta_{21}x_1 + \beta_{22}x_2 + \beta_{23}x_3 + \beta_{24}\theta_{t_0} + \beta_{25}\theta_{t_0}^2 + u_2(x, \theta_{t_0}). \tag{4.4}$$

Figure 4.3 shows a scatter plot matrix of the output $f_3(x, \theta)$ as a function of all

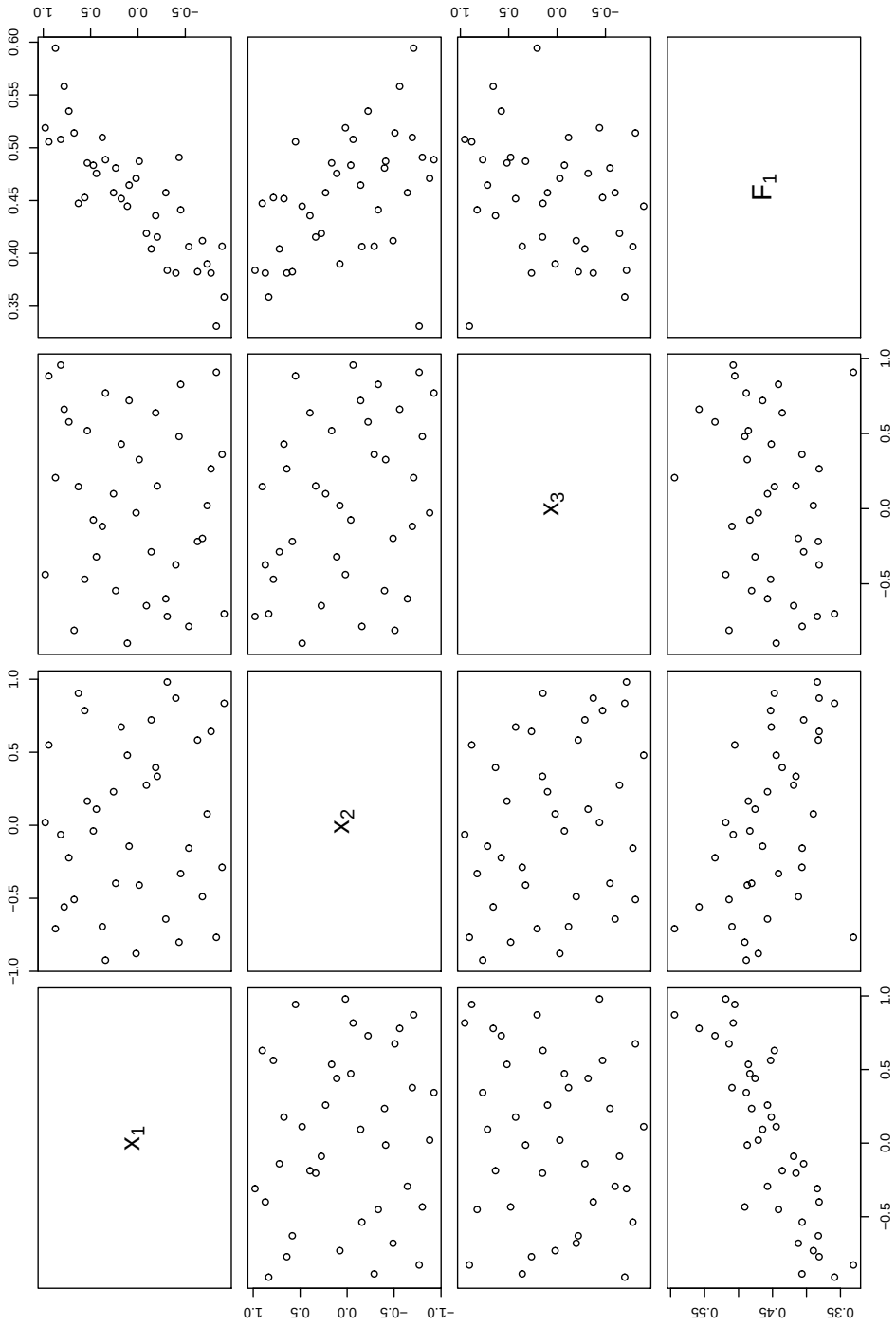

Figure 4.1: Pairs graph for the model output 1995 temperature anomaly, F_1, for different values of the model input variables x_1, x_2, and x_3.

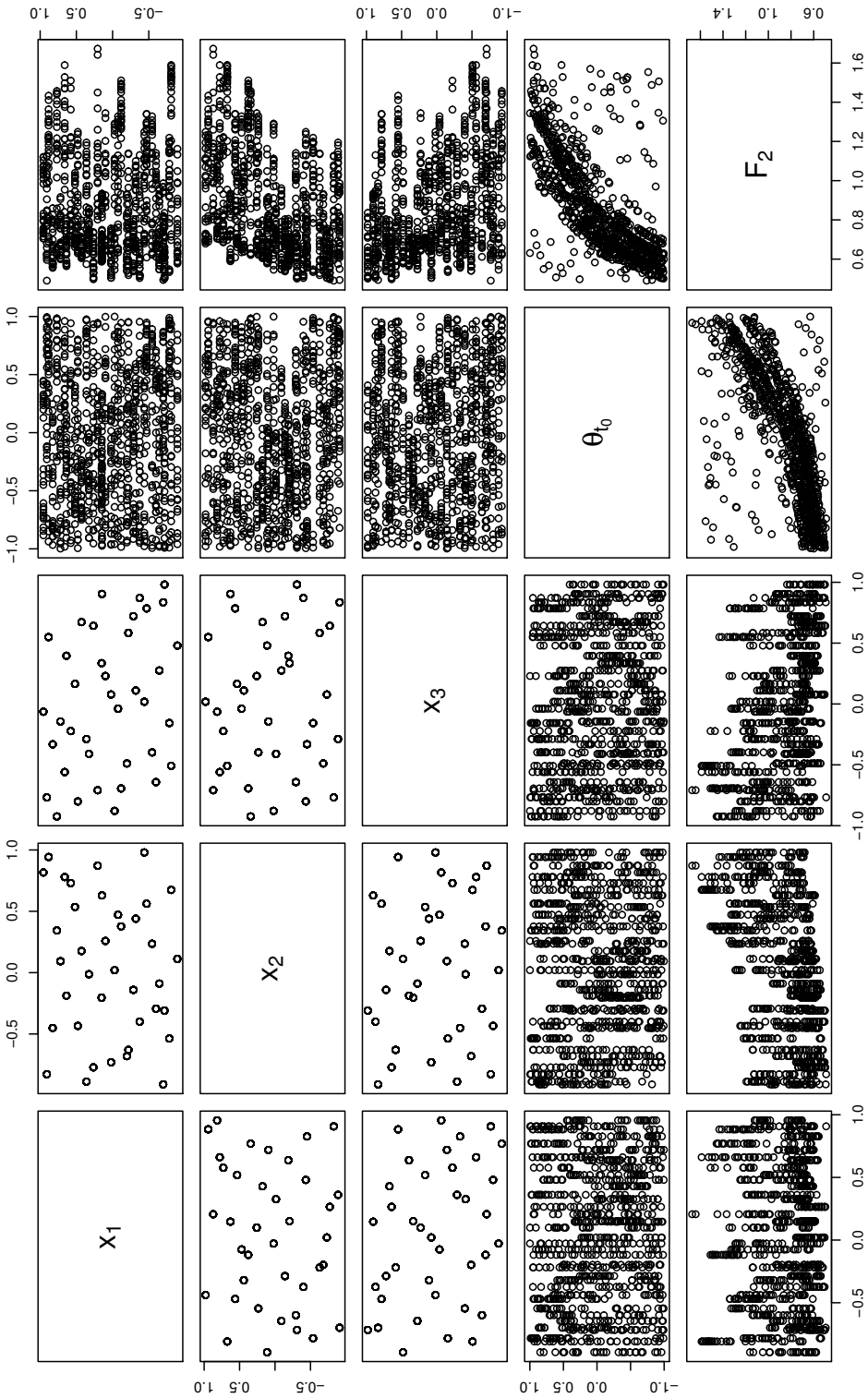

Figure 4.2: Pairs graph for the model output 2035 temperature anomaly F_2 for different values of the model input variables x_1, x_2, x_3 and the decision variable θ_{t_0}.

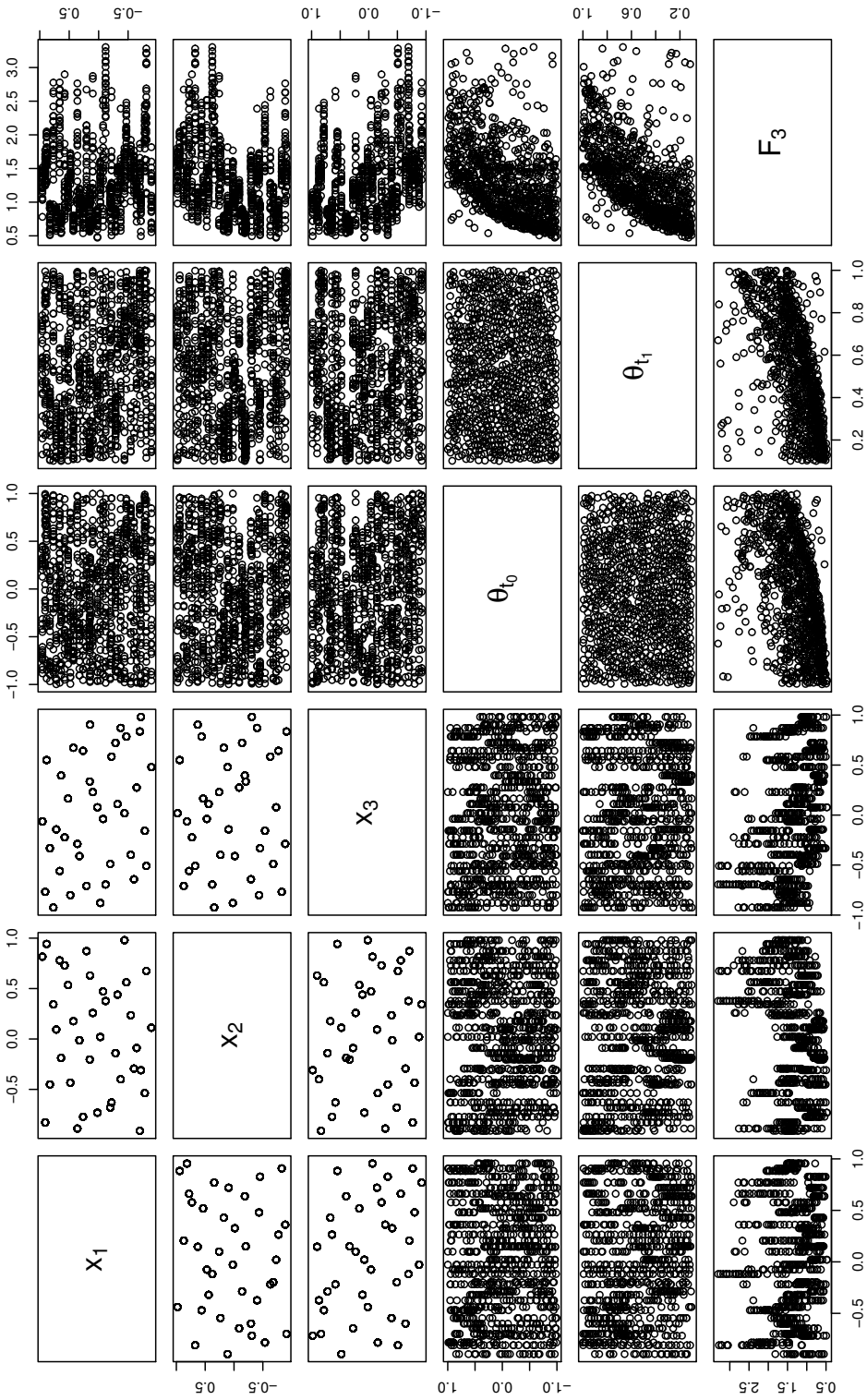

Figure 4.3: Pairs graph for the model output 2095 temperature anomaly F_3 for different values of the model input variables x_1, x_2, x_3 and the decision variables θ_{t_0} and θ_{t_1}.

of the inputs for the model. We explored fitting linear models with quadratic terms in θ_{t_0} and θ_{t_1} and found that with terms $\theta_{t_0}^2$, $\theta_{t_1}^2$ and $\theta_{t_0}\theta_{t_1}$, a separate linear term in θ_{t_1} did not change the quality of our fit. We therefore fit the model

$$f_3(x,\theta) = \beta_{31} + \beta_{32}x_1 + \beta_{33}x_2 + \beta_{34}x_3 + \beta_{35}\theta_{t_0} + \beta_{36}\theta_{t_0}^2 + \beta_{37}\theta_{t_1}^2 + \beta_{38}\theta_{t_0}\theta_{t_1} + u_2(x,\theta). \tag{4.5}$$

In a standard regression we should not leave out the linear term in θ_{t_1} because, by fitting a quadratic term, we implicitly fit a linear term. When we did include a linear term in θ_{t_1} we found that it's contribution was very small and that other coefficients did not change by very much. It was felt that the computational benefit received by not including it, outweighed any concern about it's impact, for the purposes of our illustration.

Our vector of monomials $g(x,\theta)$ is therefore

$$g(x,\theta) = (1, x_1, x_2, x_3, \theta_{t_0}, \theta_{t_0}^2, \theta_{t_1}^2, \theta_{t_0}\theta_{t_1})^T. \tag{4.6}$$

The adjusted R^2 values for each of these fits was 0.926, 0.7975 and 0.7535 respectively. Whilst the adjusted R^2 does not carry the same interpretation here as it would in a standard regression, it is a useful diagnostic tool that we may use to see if our fits are reasonable. It also gives an indication of how important our specification of the emulator residual will be to the accuracy of our final emulator. The R^2 values we have here indicate that our fitting has explained a large portion of the variation in our data.

4.3.3 Constructing a second-order prior for β

Each of the independent linear models (4.3), (4.4) and (4.5) give estimates for $E\left[\beta\right]_{ij}$ and $Var\left[\beta\right]_{ijik}$ for $i = 1,\ldots,3$ and $j,k = 1,\ldots,8$, via the usual regression output. As discussed previously, we must be careful when using output from these linear models as the usual assumptions on the residuals are not valid. We use the regression output here as part of our second order prior for β. As we have no expert knowledge to draw from, this represents a useful way of obtaining early beliefs about our computer model. We have reserved 10 spin ups (corresponding to 100 runs) in order

to adjust these beliefs, hence our final emulator will be based on a Bayes Linear fit on these.

To insure against over confidence in our prior variance for the β_{ij}s, we used leave one out diagnostics. This involved removing each data point in turn, refitting the linear model, and recording any estimate for the β_{ij}s that had changed by more than 2 standard deviations according to the values in table (4.2). We found that no estimate for the β_{ij}s changed significantly. This could suggest that we have not been confident enough in our specification. However, we wish to account for our uncertainty in unsampled areas of the design space. Therefore, having prior variances that are cautious reflects a judgement that our regression may be changed by new runs and gives our emulator flexibility.

	k=1	k=2	k=3
$E[\beta_{k1}]$	0.45	0.84	0.91
$E[\beta_{k2}]$	0.073	0.19	0.34
$E[\beta_{k3}]$	-0.044	-0.13	-0.28
$E[\beta_{k4}]$	0.0092	-0.053	-0.15
$E[\beta_{k5}]$	0	0.32	0.20
$E[\beta_{k6}]$	0	0.12	0.15
$E[\beta_{k7}]$	0	0	1.1
$E[\beta_{k8}]$	0	0	0.44

Table 4.1: Prior expectations for the regression coefficients in our emulator for C-Goldstein, rounded to 2 significant figures.

We present our prior expectation and our prior standard deviation for each β_{ij} in tables (4.1) and (4.2) respectively. Note that we have included a number of zero variances. This is because certain regressors in $g(x, \theta)$ are not present in each of the three models for the outputs. For example, terms involving θ_{t_0} or θ_{t_1} are never involved in the model for $f_1(x)$, because we are modelling historical temperatures which cannot be functions of future policies. To allow us to work with matrices in our model updating, we include these parameters but give them expectation and variance equal to zero.

	k=1	k=2	k=3
$SD(\beta_{k1})$	0.0082	0.012	0.045
$SD(\beta_{k2})$	0.015	0.015	0.041
$SD(\beta_{k3})$	0.015	0.015	0.043
$SD(\beta_{k4})$	0.015	0.016	0.045
$SD(\beta_{k5})$	0	0.015	0.095
$SD(\beta_{k6})$	0	0.028	0.079
$SD(\beta_{k7})$	0	0	0.081
$SD(\beta_{k8})$	0	0	0.16

Table 4.2: Prior standard deviations for the regression coefficients in our emulator for C-Goldstein, rounded to 2 significant figures.

The expected values of the coefficients of the decision variables in table (4.1) appear to be intuitive. We would expect these coefficients to be positive so that the more we abate (corresponding to θ decreasing) the lower the temperature becomes. We also note that the absolute value of the expectation of our model input coefficients increases with time, so that as the model is run into the future, their influence on the output becomes more pronounced. Our standard deviations for many of the coefficients, shown in table (4.2), may appear very small. However, relative to the domain of each of the inputs, the values of our expectations and the scale of the output, their size is roughly what we expect. The small variance we have on the coefficients is an indication that our residual may contain most of our uncertainty about the model output.

We note that the standard deviation of β_{14} is such that we have no reason to believe that β_{14} is non-zero. This might suggest that the variable x_3 is not active, particularly for the output corresponding to 1995. We fit on this variable and include a non-zero expectation for two reasons. Firstly, the runs that we shall be adjusting our prior emulator on may indicate that this variable is more active than it at first appears. By including this term in our emulator for 1995 temperature, we allow the runs to indicate whether or not it has a significant presence. Secondly, the means and standard deviations of the coefficients on x_3 for the outputs corresponding to

2035 and 2095 indicate that this input does have a linear relationship with the output.

In order to relate each of the three independently fitted models, we specify a correlation structure for our regression coefficients across different model outputs. We let

$$\begin{aligned} Corr\left[\beta_{ij}, \beta_{kl}\right] &= \exp\{\tau_j (t_i - t_k)^2\} \qquad & j = l \\ &= 0 & j \neq l, \end{aligned}$$

where t_i represents the actual year at index i. With this specification the amount of correlation is controlled by the τ_j's, which are chosen to be $log(0.9997)$. This choice has the desirable property that if we were fitting a regression for each year of the model output, the coefficient would not change by much year on year. The correlation between coefficients in any two adjacent years is large enough with this choice so that correlation between the coefficients of our emulator is not negligible.

4.3.4 The residual process $u(x, \theta)$

We stipulate that the form of the residual process will be the same across the decision space and that it will be uncorrelated with β. For ease of computation, we consider $u(x, \theta)$ to be a mean zero Gaussian process, separable in x and θ. We define its correlation function through

$$Cov\left[u_i(\xi_{[i]}), u_k(\xi'_{[k]})\right] = V_{ik} * \sigma_i \sigma_k * \exp\{-(\xi_{[m]} - \xi'_{[m]})\Lambda_{[m]}(\xi_{[m]} - \xi'_{[m]})^T\}, \quad (4.7)$$

where $\xi_{[1]} = (x_1, x_2, x_3)$, $\xi_{[2]} = (x_1, x_2, x_3, \theta_1)$, $\xi_{[3]} = (x_1, x_2, x_3, \theta_1, \theta_2)$, i and k index the output of the simulator, $m = \min(i, k)$ and Λ is a diagonal matrix containing the inverse squares of the correlation lengths. $\Lambda_{[m]}$ indicates a submatrix of Λ, appropriate to m so that $\Lambda_{[m]}$ contains only those correlation lengths for variables in $\xi_{[m]}$. The matrix V controls correlation between the output residuals and the vector σ indicates the magnitude of residual standard deviation on each output. We define $\xi_{[m]}$ in this way so that only the distance between inputs that influence the value of both $u_i(x, \theta)$ and $u_k(x, \theta)$, contribute to their covariance. So, for example, if considering the covariance between $u_1(x, \theta)$ and $u_3(x, \theta)$, we do not want the value

of θ, which does not affect $u_1(x, \theta)$, to contribute to the covariance between it and another quantity depending on θ. This notation, therefore, indicates that there is only correlation in the residual induced by shared inputs of each function.

If fitting an emulator using data alone, one way to fix the parameters of the residual process is via variogram fitting. In our case, this would be inappropriate as we only have 38 distinct points in our design for the model input space, and yet we have 11 parameters to fit (although some of these, depending only on θ, may be more attainable). Instead of using variograms, we use a combination of the standard deviation of the residuals from our least-squares fitting, leave one out and graphical diagnostics, as well as our own judgements.

To simplify our formulation, we judged that temporal correlation would be adequately handled by the covariance on the regression coefficients and hence set V to be the 3×3 identity matrix. In a more serious analysis we would have to consider this choice more carefully, either using expert judgement or a correlation structure similar to that we imposed on the regression coefficients.

In order to choose the elements of σ, we relied heavily on leave one out diagnostics. We may have been tempted to set σ_1, σ_2 and σ_3 to be the residual standard errors from our three fits. The leave one out plots of figures 4.4, 4.5, 4.6, 4.7 and 4.8 show why this would be a mistake. In each of these pictures, the red dotted lines represent 2 standard deviations of the residuals, either side of zero, according to our original regression fits. The plots show some points 5 or more standard deviations away from where our model predicts. With such small values of σ_1, σ_2 and σ_3, our model would struggle to cope with surprising observations and would not even be suitable for those we have made now. We therefore choose σ_1, σ_2 and σ_3 in accordance with these plots so that the extreme observations are consistent with our variance specification. We set

$$\sigma = (0.05, 0.25, 0.7)^T$$

so that the variance on the residual also reflects our beliefs about how far new observations might deviate from our regression surface.

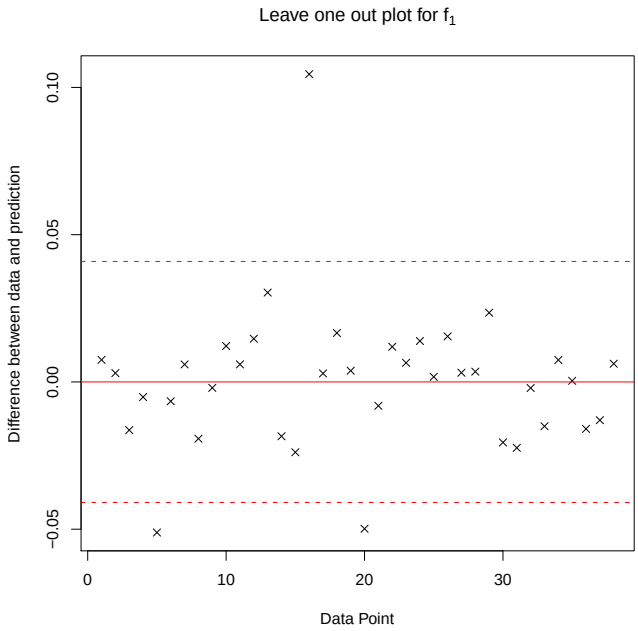

Figure 4.4: Leave one out diagnostic graph used to fix σ_1. The dotted red lines represent ± 2 standard deviations with respect to the original linear model. Each of the points represents the difference between the data point left out and the prediction for it using a linear model fitted to the rest of the data.

Fixing the correlation lengths

To choose the correlation lengths on each variable we use an heuristic designed to give values with sensible properties. Whilst we do not have enough data to attempt variogram fitting, even if we did, using an heuristic argument to choose Λ may be more appropriate. Variogram fitting assumes we have a sample from a Gaussian process and is a technique from Geostatistics used to estimate the parameters. It is important, firstly, to note that we are not sampling from a Gaussian process and that the computer model is, in fact, deterministic. Secondly, if using variogram methods as a computational tool in order to estimate values of the parameters, we must be wary of the fact that the mean, variance and correlation length in our statistical model are confounded.

Variogram fitting involves using numerical optimization to fit the 'best' mean, variance and correlation parameters for a Gaussian process assuming the data is a sample from that process. Figure 4.9 shows the way in which the confounding of

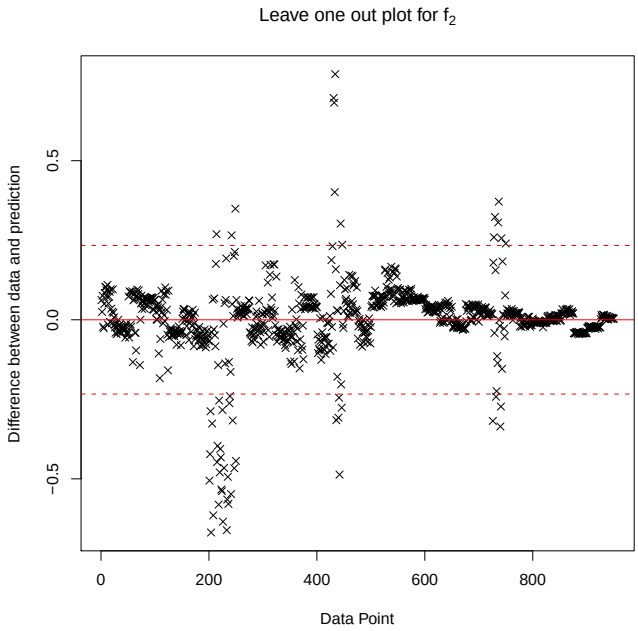

Figure 4.5: Leave one out diagnostic graph used to fix σ_2. The dotted red lines represent ± 2 standard deviations with respect to the original linear model.

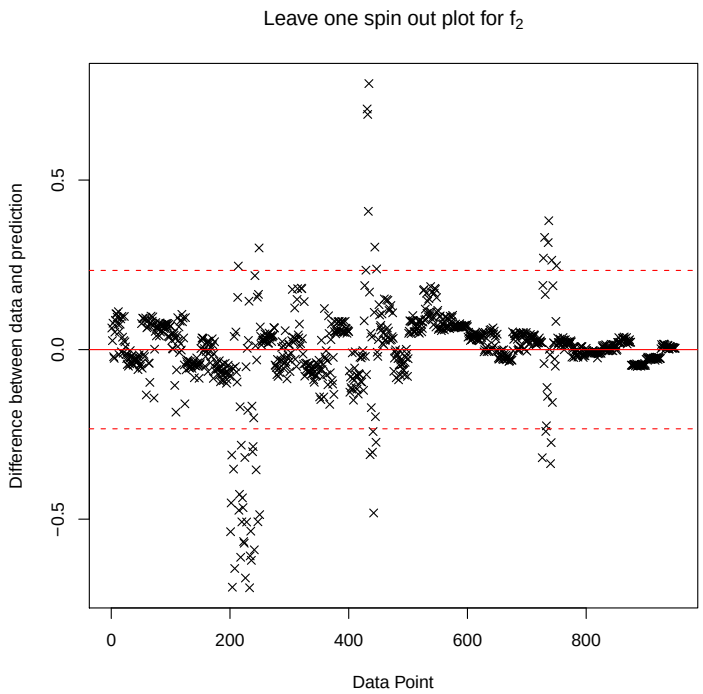

Figure 4.6: Leave one spin up out diagnostic graph used to fix σ_2. The dotted red lines represent ± 2 standard deviations with respect to the original linear model.

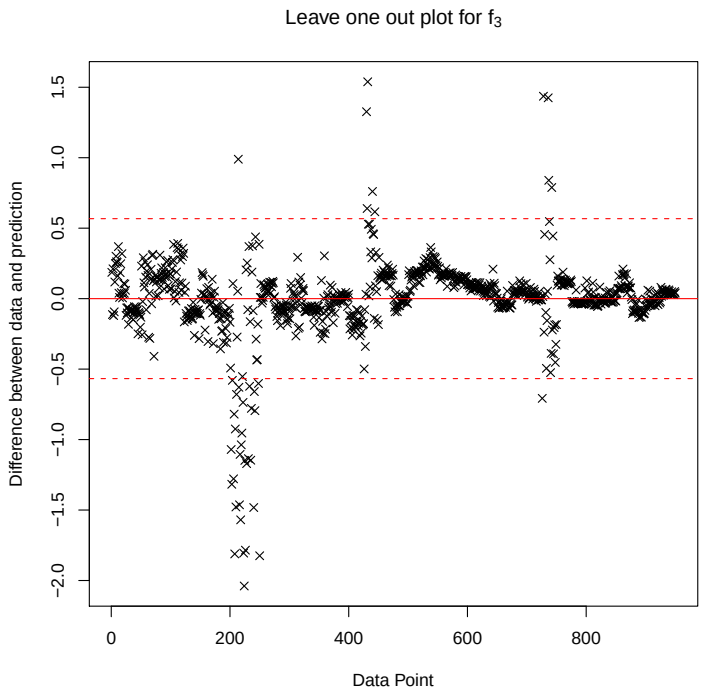

Figure 4.7: Leave one out diagnostic graph used to fix σ_3. The dotted red lines represent ± 2 standard deviations with respect to the original linear model.

these three things can manifest itself in an emulator. Our goal is to build an emulator that fits a smooth curve through all of the points in the picture. To do this, we fit a mean function, a variance and some correlation parameters. Two possible mean functions are shown in in the picture. The one going through the points is an accurate reflection of the sort of global regression we would like to fit. We then use a Gaussian process with an appropriate correlation length and a relatively small variance, and our emulator would be a smooth curve going through all of the points. To obtain exactly the same curve we could start with the mean function represented by the blue line, choose a large variance and choose long correlation lengths. Away from any observed points, our emulator reverts back to the mean at a rate determined by the correlation length. Therefore, in this illustration the second example, whilst giving the same quality of fit to the points, would be much worse globally. By using an optimization technique such as variogram fitting to fit the parameters to our data, we are in danger of accidentally ending up in this second situation. Whilst this example may be extreme, it shows the way that each

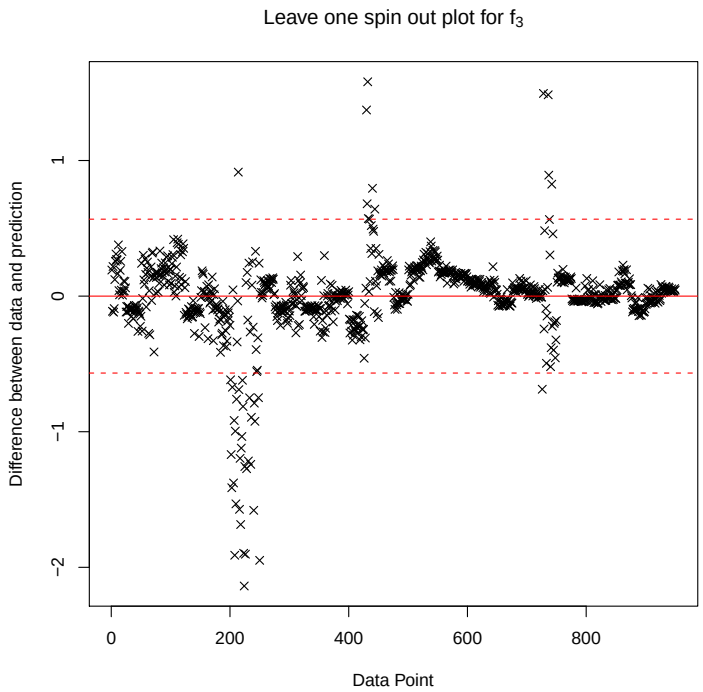

Figure 4.8: Leave one spin up out diagnostic graph used to fix σ_3. The dotted red lines represent ± 2 standard deviations with respect to the original linear model.

of the parameters we are attempting to fit is confounded. Therefore, it may be better to use an heuristic to choose the correlation lengths even with large amounts of data. That way we can be sure that our choice of correlation length is not simply an artifact of the modelling we have already performed, which does not reflect our beliefs about how much correlation may be in the residual.

We imagine a Taylor expansion of the complex computer model function in each of its inputs that we have varied. As we have fitted a polynomial regression to the function, then if we consider that polynomial to capture behaviour of its order well we can assume that the residual demonstrates behaviour one order higher than the fitted regression components. For example, we have fitted linear terms in each of the three model input variables and, by assuming that we have captured this behaviour well, the residual should have quadratic and higher order behaviour in all three of these inputs.

Having decided what order effects we wish to capture via the residual, we then address the problem of correlation length by considering the number of local maxima

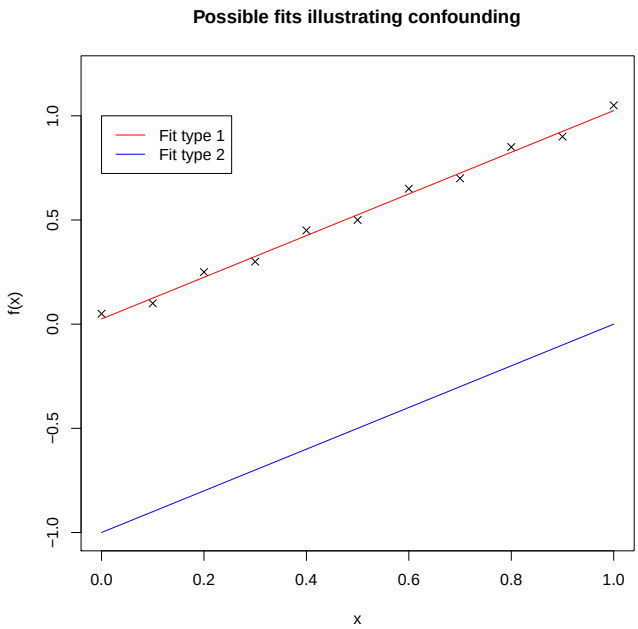

Figure 4.9: This picture shows two potential fits (represented by the lines) to a set of points. We argue in the text that with certain fitted variances and correlation parameters we could build two emulators that both go through the points in exactly the same way.

or minima a general polynomial of that order would have. We then conduct a thought experiment to decide how informative we wish a point at one of these extreme values to be for a point at its nearest neighbouring extreme value. We illustrate this using the example. Choosing the variable x_1, we decide that the order of the residual variation should be quadratic.

Figure 4.10 shows a quadratic shaped curve that we will use to help visualise the thought experiment. We are considering how two points might be correlated, so we assume they both have the same values of x_2 and x_3 and that we would like a point at one of the end points to be "informative" for a point half way to the turning point (the intersection of the curve with one of the red lines), but not informative for the location of the turning point. Here we decide that "informative"

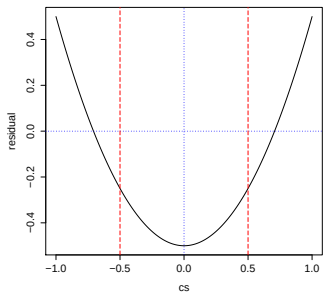

Figure 4.10: Typical quadratic curve

equates to having a correlation of e^{-1}, and so we set the correlation length for x_1 to be 0.5. Although this choice may seem somewhat arbitrary, it obtains the desired sensible properties referred to whilst not putting in too much correlation. A similar heuristic to the one described in which the low order excluded effects are used to choose a correlation length, was used by Goldstein and Rougier in [39] and by Craig et al in [19]. Using this heuristic we set the diagonal of Λ to be $(4, 4, 4, 9, 9)^T$

4.3.5 Completing our emulation

Our emulator is completed by adjusting our prior judgements for β and $u(x, \theta)$ by the 100 runs we set aside in the beginning. We present the R code used to perform this adjustment in sections D.1.1 and D.1.2 of the appendices. At this stage in our analysis we compute the adjusted expectation and variance of β using the Bayes Linear equations (2.9) and (2.10). The rest of the adjustment happens as part of the forecasting calculation and the R code for this is presented in sections D.1.3 and D.1.4 of appendix D. It is appropriate at this point to comment on the performance and interpretability of our emulator for C-GOLDSTEIN.

We check the performance of our emulator using leave one out diagnostics. To produce leave one out diagnostics for our emulator, we remove each point in our data then adjust the prior emulator by the remaining 99 data values. We use the adjusted mean and variance of this emulator to calculate a prediction for the missing data point and compute the standardized distance between the data and the prediction. We perform this test for each of the three outputs individually (although our adjustments are multivariate) and plot the results in figure 4.11, figure 4.12 and figure 4.13. Note that we remove each spin up and adjust on the remaining 9 spin ups when testing the validity of our emulator for $f_1(x)$.

The diagnostic plot in figure 4.11 may worry us a little in that we have one observation almost 3 standard deviations away from where it was predicted to be. As a rule of thumb, anything more than 3 standard deviations away may cause us to re-evaluate our emulation. In practice we would want to extend the range of x values used to evaluate the model and adjust our emulators, as ten is very few and it is difficult to interpret whether or not our emulator is performing well or poorly

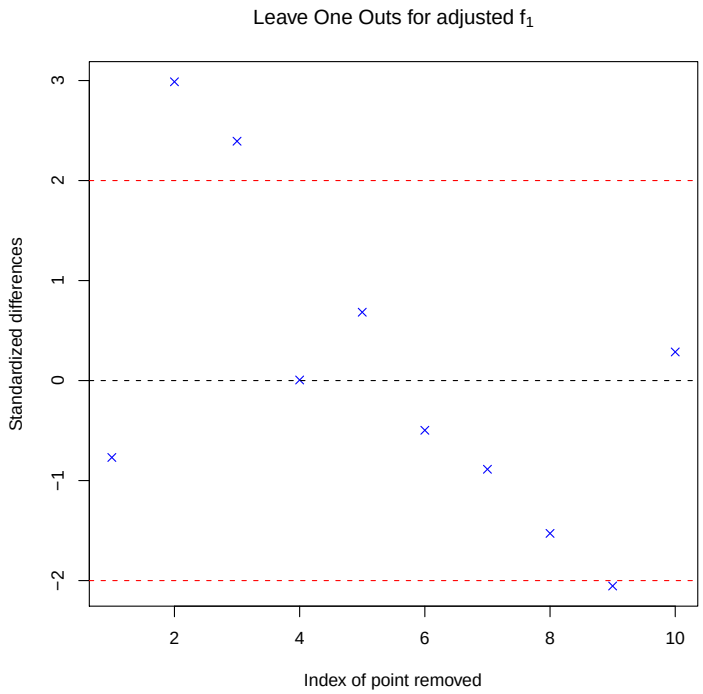

Figure 4.11: Leave one out diagnostic graph used to check the performance of our adjusted emulator for $f_1(x)$.

from only ten runs. We may also be concerned by the apparent linearity in the leave one out plot in figure 4.11. There is no reason that the index of the points we are removing should be a factor in the quality of our fits, however, it may be that our designs had some ordering that we were not aware of. In a more careful study of this model we could explore this.

The other leave one out plots in figure 4.12 and figure 4.13 show that each of our predictions are doing very well (almost better than expected) for the ten spin ups we have. This is possibly because each prediction, for any removed point, is made with its value of $f_1(x)$ known and, through our correlation structure on the β, we learn something about the missing point from its spin up value. It is important to note that we have no information about how well we are doing away from the ten spin ups we have. We can, however, be reasonably confident that we have captured the important features of the simulator with our emulator, because for each spin up that we have seen, the relationship between future temperatures and $(\theta_{t_0}, \theta_{t_1})$ appears roughly the same. We provide an example of this in figure 4.14. This

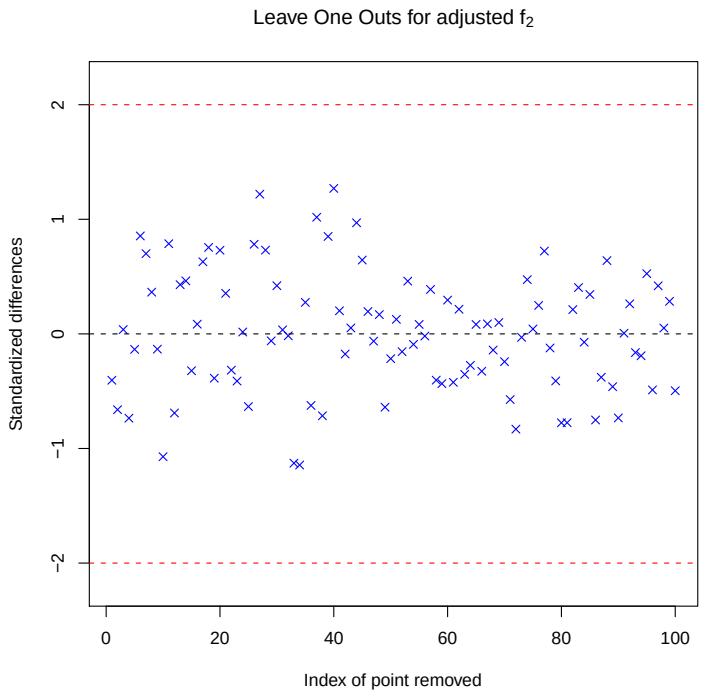

Figure 4.12: Leave one out diagnostic graph used to check the performance of our adjusted emulator for $f_2(x, \theta)$.

picture shows the relationship between our model output for 2035 and θ_{t_0} with each spin up highlighted. We see that each spin up appears to define a quadratic curve in θ_{t_0} with approximately the same shape each time.

The three leave one out plots do not contain anything overly alarming and, for the purpose of this illustrative example, we can be fairly confident that our emulator will be fit for purpose. In a case study, we would consider many more diagnostics and explore alternative correlation lengths as well as perhaps the sensitivity of other aspects of our prior specification. We might consider using cross validation and, perhaps more importantly, would consider more carefully how the first 38 spin ups were used.

It is important to note that our emulator only interpolates the 100 runs we adjusted on. Our emulator therefore does not interpolate the function at every point at which we have observed it, although we know that our prior model was consistent with those observations. We may easily obtain an interpolator by adjusting our emulator by the 950 runs we used to build the prior. Double counting of the data

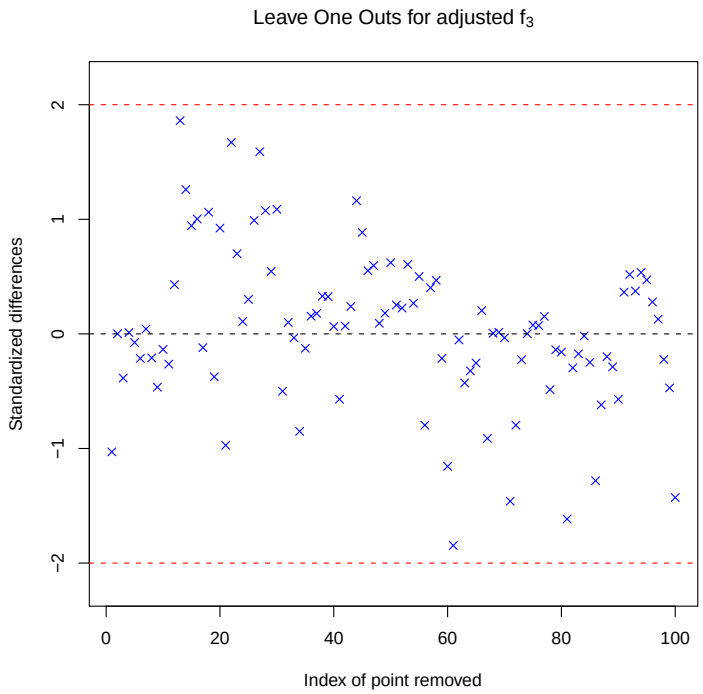

Figure 4.13: Leave one out diagnostic graph used to check the performance of our adjusted emulator for $f_3(x, \theta)$.

in this way, however, would lead to over confidence in the performance of our model away from the data points. We therefore view the option of having an emulator built using knowledge gleaned from an initial 950 runs and interpolating a further hundred, as being the more favourable choice. The ideal scenario would involve using expert knowledge to build our priors and to adjust by all of the runs.

We provide further details about our final emulator for C-GOLDSTEIN in appendix E.1.

4.4 DICE as a loss function

Although DICE is a complex loss model, in order to illustrate our methods, we intend to treat it as a function of y and θ only. We must therefore fix each of DICE's input parameters. We set most of these at their default values, specified in the downloadable spreadsheet version of DICE-99 available from [79]. A number of parameters were changed to values that gave an 'interesting' setting of DICE,

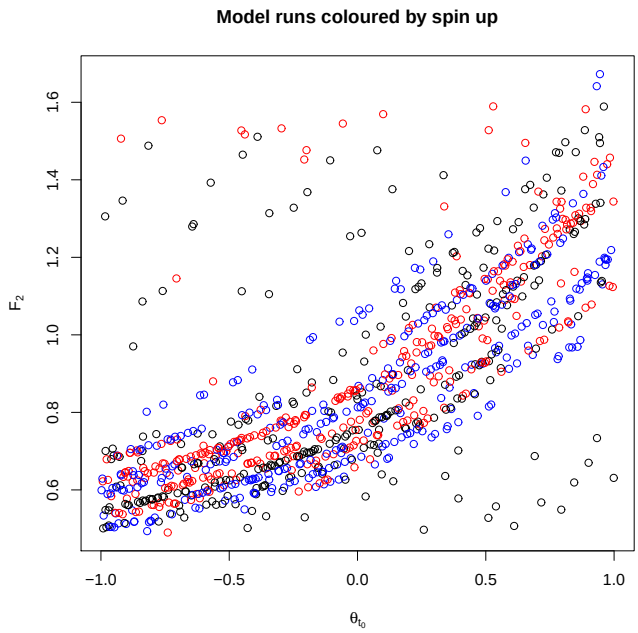

Figure 4.14: A plot of the model data for 2035 against θ_{t_0}. We colour each spin up black, red, and blue alternately. This plot demonstrates that the relationship between the model output and θ_{t_0} is roughly the same for each spin up.

as initial exploration of the default setting showed that we should always make no reduction to CO_2 emissions today, whatever the consequences for the future were.

The evolution of abatement cost in DICE is controlled by the function

$$b_1(t) = \frac{b_1(t-1)}{1 + g^b(0)exp(-\delta_b t)}.$$

We change the default values of $b_1(0)$ and $g^b(0)$ from 0.03 and -0.08 to 0.005 and 0.25 respectively. The new value of $g^b(0)$ provides a larger initial damping of $b_1(t)$ and makes abatement cheaper today. The smaller $b_1(0)$ values has the effect of making overall abatement cheaper. We do this so that the cost of abatement does not prohibit ever choosing to manage CO_2 emissions and to provide an interesting example that illustrates our methods. We also fix our preferences for the wealth of future generations by setting $r = 0.1$ in equation (4.1).

Figure 4.15 and figure 4.16 represent a visualization of DICE output for different inputs. From figure 4.15 we see that future temperature has a dominating effect on the consequences we face for any particular policy. However, we know that the policies will themselves drive future temperature and from figure 4.16 we can see

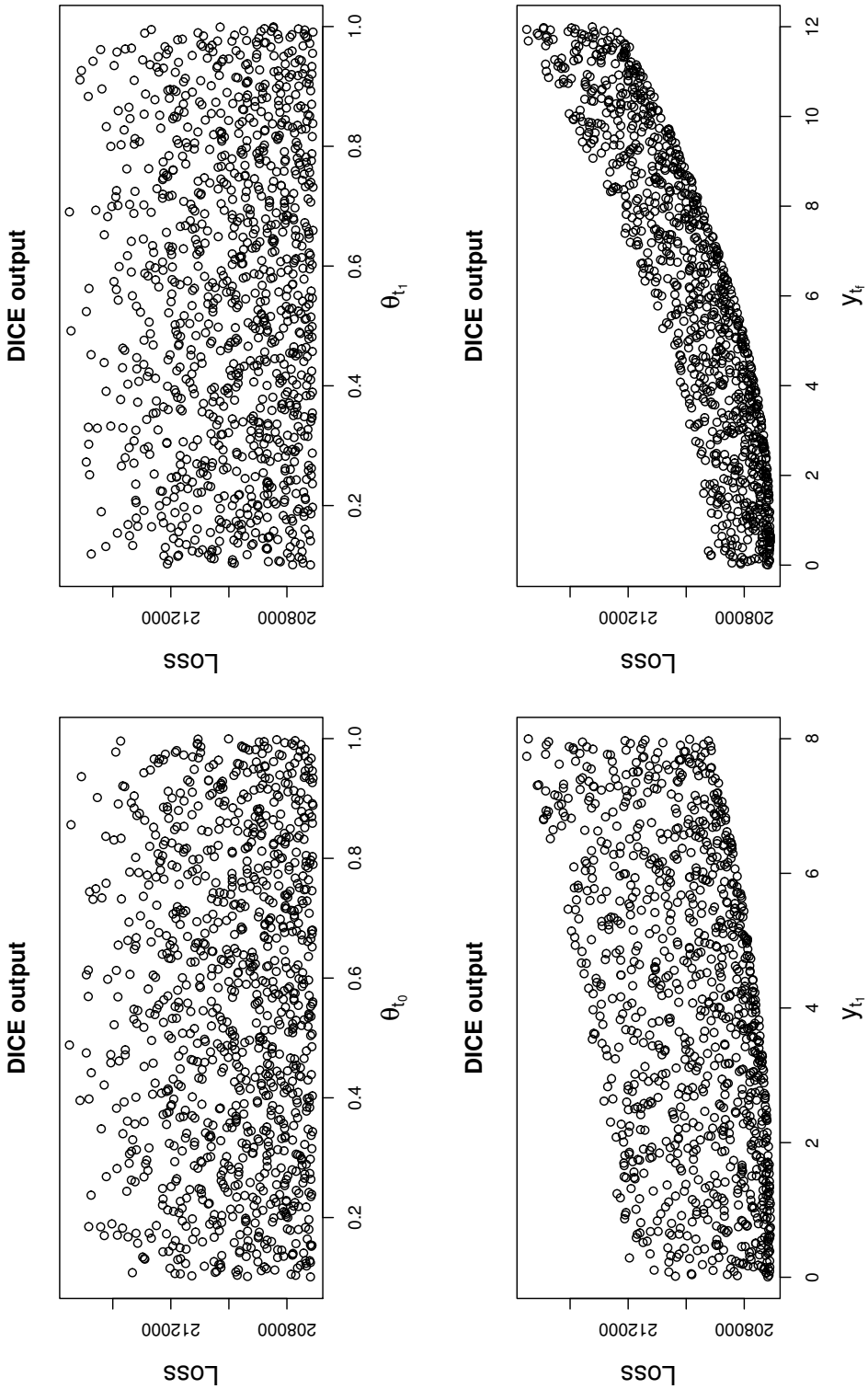

Figure 4.15: Scatter plots of DICE output for a Latin Hypercube Design on the policies and temperatures. Loss here is evaluated as negative utility.

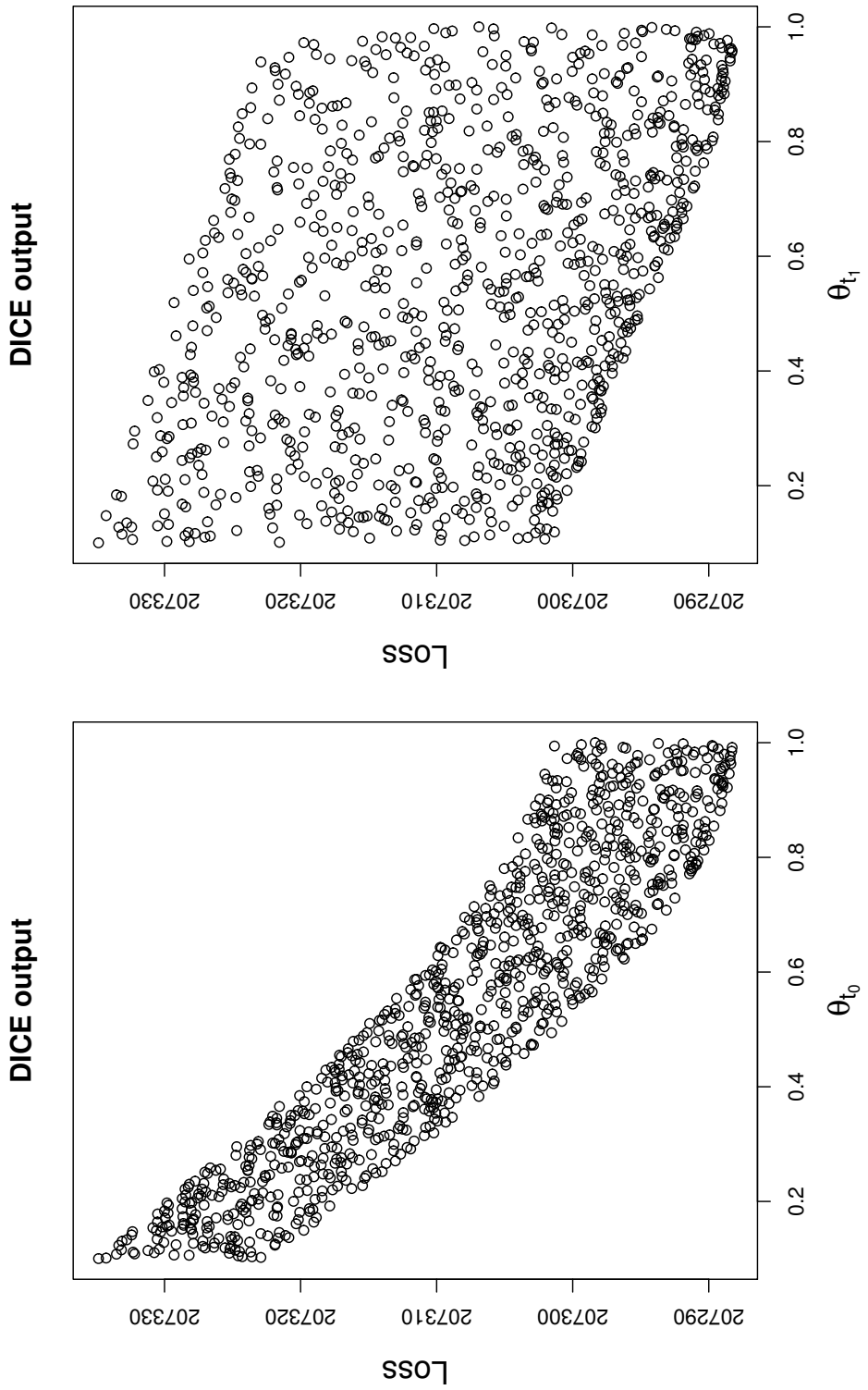

Figure 4.16: Two scatter plots of DICE output for different policies with the temperature input fixed. Loss here is evaluated as negative utility.

that for given future temperatures we would like to have made as little reduction to emissions today as possible. We explore the sensitivity of our methods to the choice of r in section 4.7.4.

4.5 Completing the preliminaries

Before we begin our Sequential Emulation of the decision tree we must complete preliminary steps P2, P3, P4 and P5.

4.5.1 Obtaining forecasts

We begin at P2 by writing down our beliefs about x^*, $\eta(\theta)$, and e. We choose independent uniform prior distributions for x_1^*, x_2^*, and x_3^*, to reflect ignorance and to facilitate easier numerical integration. Whilst this serves the purpose of illustrating our methods, a more careful description of $p(x^*)$, such as that undertaken by Rougier and Kern [98], would usually be required.

We specified the variance of mean-zero observation error e in section 4.1 on page 93. We gave the discrepancy mean zero and decided to make it independent of θ_{t_0} and θ_{t_1}. To construct its variance matrix we consulted Peter Challenor from the National Oceanography Centre in Southampton, who has experience with C-GOLDSTEIN (see Challenor and Marsh [16]). Peter gave standard deviations on η_{t_0}, η_{t_1}, and η_{t_2} as 0.1, 0.5 and 1 respectively. He also specified correlations

$$Corr\,[\eta_1, \eta_2] = \sqrt{0.5}, \quad Corr\,[\eta_1, \eta_3] = \sqrt{0.1}, \quad Corr\,[\eta_2, \eta_3] = \sqrt{0.25}.$$

To obtain a decision-dependent forecast, we must compute the integrals (2.22), (2.23) and (2.24) for any given decision. Let

$$h(x) = (1, x_1, x_2, x_3, 1, 1, 1, 1)^T, \qquad k(\theta) = (1, 1, 1, 1, \theta_{t_0}, \theta_{t_0}^2, \theta_{t_1}^2, \theta_{t_0}\theta_{t_1})^T$$

and

$$r^x(|x - x'|) = \exp\{-(x - x')\Lambda_x(x - x')^T\}$$
$$r_{ik}^{\theta}(|\theta - \theta'|) = \exp\{-(\theta_{[m]} - \theta'_{[m]})\Lambda_\theta(\theta_{[m]} - \theta'_{[m]})^T\} \qquad m > 1$$

with

$$\Lambda = \begin{pmatrix} \Lambda_x & 0 \\ 0 & \Lambda_\theta \end{pmatrix},$$

$m = min(i,k)$, $\theta_{[2]} = \theta_{t_0}$, $\theta_{[3]} = (\theta_{t_0}, \theta_{t_1})$, and $r^\theta_{ik}(|\theta - \theta'|) = 1$ if $m = 1$. Then

$$g_i(x,\theta) = h_i(x) * k_i(\theta)$$

and

$$Cov\left[u_i(\xi_{[i]}), u_k(\xi_{[k]})'\right] = V_{ik} * \sigma_i\sigma_k * r^\theta_{ik}(|\theta - \theta'|)r^x(|x - x'|)$$

(where $\xi_{[j]}$ is defined on page 106 for $j = 1, \ldots, 3$). Hence we have separability in both $g(x,\theta)$ and in our residual covariance function. We therefore only have to integrate any $x-$dependent quantities once, before using (2.27), (2.28), (2.29), (2.30), and (2.31) to compute $E\left[y(\theta)\right]$ and $Var\left[y(\theta)\right]$ for any θ, without integrating again.

The only integrals we must solve in order to produce decision-dependent forecasts then, are

$$\int_{-1}^{1} h(x^*)p(x^*)dx^*, \qquad \int_{-1}^{1} h(x^*)h(x^*)^T p(x^*)dx^*,$$
$$\int_{-1}^{1} r^x(|x^* - \Omega|)p(x^*)dx^*, \qquad \int_{-1}^{1} r^x(|x^* - \Omega|)r^x(|x^* - \Omega|)^T p(x^*)dx^*,$$
$$\int_{-1}^{1} h(x^*)r^x(|x^* - \Omega|)p(x^*)dx^*.$$

We can solve each of these analytically; using Normal distribution theory to obtain the integrals involving $r^x(|x^* - \Omega|)$. We detail these calculations in Appendix B. Once we have performed the necessary integrations, our decision-dependent forecasts require no further numerical integration. Our 'coarse' and 'accurate' methods of forecasting, as required in step P3, are the same because exact forecasts are obtained that are appropriate to our method of emulation.

4.5.2 Characterizing the required distributions

Preliminary step P4 requires us to specify how forecasts, such as the pair $E_{z^m}\left[y;\theta^m\right]$ and $Var_{z^m}\left[y;\theta^m\right]$, will characterize the various distributions required to compute the different expected losses on our decision tree. In a case study, this step would

be critically important and would receive a lot more attention. Indeed, the idea of using partial beliefs to characterize probability distributions; the interpretations of any probability statements implied by those distributions; and the sensitivity of any drawn conclusions to the particular characterization (or any perturbations in the chosen distribution); would form an interesting and worthy area of research in its own right. We do not address these questions in this thesis and choose our characterization for convenience only.

We must determine how our forecasts will characterize $p(y|z_{t_1}, z_{t_0}, \theta_{t_0}, \theta_{t_1})$ and $p(z_{t_1}|z_{t_0}, \theta_{t_0})$. We let both of these distributions have a log-normal form with mean and variance determined by the appropriate forecast. We make this choice because the multivariate log-normal distribution is easily sampled from, and because its support is strictly positive. This amounts to a judgement that the global mean temperature will not decrease below the 1900 temperature within this century. An undesirable feature of this choice is the unrealistically high probability given to extremely large temperature increases. Under these distributions we often sample temperature increases of 6° or 7° this century! In a serious attempt to provide decision support to policy makers this would be unacceptable, however, our illustration of the key features of the methodology is well served by using easily sampled quantities.

We therefore fix $p(y|z_{t_1}, z_{t_0}, \theta_{t_0}, \theta_{t_1})$ to be multivariate log-normal with mean $E_{z_{t_1}, z_{t_0}}[y; \theta_{t_0}, \theta_{t_1}]$ and variance $Var_{z_{t_1}, z_{t_0}}[y; \theta_{t_0}, \theta_{t_1}]$ for any θ_{t_0}, θ_{t_1}, and z_{t_1}. We also let $p(z_{t_1}|z_{t_0}, \theta_{t_0})$ have a log-normal distribution with mean $E_{z_{t_0}}[z_{t_1}; \theta_{t_0}]$ and variance $Var_{z_{t_0}}[z_{t_1}; \theta_{t_0}]$ for any θ_{t_0}.

4.5.3 Coarse and accurate evaluations

When integrating with respect to y, as we must when emulating $A(\theta^m, z^m)$ and $B^1_{\lambda^1}(\theta_{t_0}, z^m)$, we use Monte Carlo methods for both coarse and accurate numerical integrations. Our coarse evaluations use 1000 samples from the appropriate distribution and evaluate the mean of the loss function over those samples. Our accurate evaluations compute this mean over $100,000$ samples. When integrating with respect to z_{t_1}, as we must when evaluating $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$, we use 1000 and $10,000$ samples

for coarse and accurate integrations respectively. We feel we are able to make fewer samples for the accurate calculation as we are integrating a one-dimensional function as opposed to a three-dimensional one.

It is worth commenting that numerical Quadrature (see, for example, Davis and Rabinowitz [25] or Evans [30]) would allow us to obtain more accurate evaluations of each integral. It was felt, however, that in a serious attempt at the problem the dimensions of each integration would be such that Quadrature would be infeasible. Illustration of the method would therefore not benefit from the extra pain involved in deriving a Quadrature rule for integrating with respect to the log-normal weight function over three dimensions.

This final judgement completes the preliminary steps required before beginning our Sequential Emulation of the decision tree. We now follow the Sequential Emulation algorithm, beginning at step S1.

4.6 Sequential Emulation

4.6.1 Designs

To build each of our emulators, we require a large set of coarse evaluations and a number of accurate evaluations made at locations where we also have coarse data. For each of the three emulators required, we use Latin Hypercube designs to choose the location of our evaluations. Our emulator for $A(\theta^m, z^m)$ will be based on coarse evaluations at each point of a 1000 point Latin Hypercube in θ_{t_0}, θ_{t_1} and z_{t_1}, with the accurate data collected over a similar 200 point Latin Hypercube. Note that we obtain the coarse evaluation at each of these 200 points. Our emulator for $B^1_{\lambda^1}(\theta_{t_0}, z^m)$ will be based on a 1000 point Latin Hypercube in θ_{t_0} and z_{t_1} for the coarse data, and a 200 point Latin Hypercube for the accurate. Finally, our emulator for $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$ is based on a 1000 point Latin Hypercube in θ_{t_0} for the coarse data, and a 100 point Latin Hypercube for the accurate.

The lognormal form for $p(z_{t_1}|z_{t_0}, \theta_{t_0})$ means that when sampling from this distribution we can get large values of z_{t_1}. We must then evaluate $\tilde{E}\left[B^1_{\lambda^1}(\theta_{t_0}, z^m)\right]$ using our emulator for $B^1_{\lambda^1}(\theta_{t_0}, z^m)$. Therefore, our emulators must be accurate for these

values of z_{t_1}, indicating that our designs should contain appropriately large values of this quantity. Based on generating samples from $p(z_{t_1}|z_{t_0}, \theta_{t_0})$ for typical forecast values, we generate design points for z_{t_1} on the range $(0, 8)$.

4.6.2 Building the coarse models

The loss values we obtain from DICE for any θ_{t_0}, θ_{t_1}, and z_{t_1} are order 10^5 values. As utility is invariant to positive linear transformations, we rescale our coarse and accurate evaluations to the range $[0, 10]$. Each of our coarse emulators will be built using saturated linear models in the relevant inputs. We use the term 'saturated linear model' to refer to a model containing all feasible terms of any order less than or equal to the order of the fitted model. For example, suppose we are fitting a model in variables a, b and c, then the saturated linear model of order two must contain all of the terms a, b, c, ab, ac, bc, a^2, b^2, and c^2. We use saturated models so that the expectations of our accurate emulators can be linearly transformed back onto the DICE scale if need be.

We fit linear models to the coarse data as part of an emulation tool. The idea is to use linear fitting to choose the order of the regression surface and to fix our choices of the $g(\cdot)$ and β in (3.11). We then make judgements for each of the remaining quantities required for multi-level emulation of the accurate model and adjust by the accurate evaluations. Although we are fitting linear models to the coarse data at each step, and although we use some of the output from these regressions, it is important to note that we do not believe all of the usual regression assumptions and as such do not trust all of the diagnostic output from any regression. We treat the linear models as least squares fits and use residual plots to make judgements about the most appropriate model form and about the correlated and uncorrelated components of the coarse model residual. The output we take forward is used as part of a prior model for our accurate evaluations of the required integrals. This prior is then updated by accurate evaluations and our emulator for the integral is based on this update.

Before addressing S1, it is necessary to discuss how we treat the coarse residuals. Each emulator that we shall build when performing the algorithm is a function of a

subset of the variables $(\theta_{t_0}, z_{t_1}, \theta_{t_1})$. Denote an appropriate subset of these variables ζ, so that, for example, if we are addressing S1, $\zeta = (\theta_{t_0}, z_{t_1}, \theta_{t_1})$. From (3.11), we have

$$\pi^c(\zeta) = \beta_j g_j(\zeta) + \varepsilon^c(\zeta) + \delta^c(\zeta),$$

where $\pi^c(\zeta)$ represents the coarse evaluation of the integral we are emulating (so at step S1 this is $A(\theta^1, z^1)$), the first term represents the regression surface we shall fit using the coarse data, $\varepsilon^c(\zeta)$ is a correlated residual process representing behaviour not captured by the global regression, and $\delta^c(\zeta)$ is uncorrelated error. We let $\varepsilon^c(\zeta)$ have a Gaussian correlation function so that, for any ζ,

$$Cov\left[\varepsilon^c(\zeta), \varepsilon^c(\zeta')\right] = \sigma^2_{\varepsilon^c} exp\{(\zeta - \zeta')\Lambda_{\varepsilon^c}(\zeta - \zeta')^T\},$$

where Λ_{ε^c} is the diagonal matrix of correlation parameters appropriate to the coarse calculation and σ_{ε^c} is the standard deviation of the correlated residual. The correlation lengths are chosen using the same heuristic as discussed in section 4.3.4. However, we multiply the correlation lengths by $\sqrt{10}$ to allow more correlation into this component of the residual. This choice was based on early tests of the methodology and leave one out diagnostics. Leave one out diagnostics are very computationally expensive here so that our choice of correlation length was based on a comparison of only a handful of different candidates. Multiplying the correlation lengths derived from our heuristic by $\sqrt{10}$ gave a better fit than the original values. We do not explore the issue of correlation length further here as we do not feel it will be crucial to the performance of our emulators. We feel that the correlated component of the residual contributes only a small amount to the emulators we have built. We specify σ_{ε^c} and $\sigma_{\delta^c} = SD\left[\delta^c\right]$ by analysing the residuals from individual fits. We shall discuss this in more detail in the context of our example in section 4.6.3.

From equation (3.12) we have

$$\pi^a(\zeta) = \rho\beta_j g_j(\zeta) + \gamma\varepsilon^c(\zeta) + \varepsilon^a(\zeta) + \delta^a(\zeta)$$

where $\pi^a(\zeta)$ represents the accurate evaluation of the integral we are emulating, ρ and γ are scalar quantities we must specify beliefs about, $\varepsilon^a(\zeta)$ is the correlated component of the accurate residual, and $\delta^a(\zeta)$ is uncorrelated error. Our residual

process for any of the accurate models, $\varepsilon^a(\zeta)$, will have the same form of covariance function as that for the coarse, and will have the same correlation lengths. We choose σ_{ε^a} and σ_{δ^a}, the standard deviations of the correlated and uncorrelated components of the accurate residual respectively, based on residuals calculated from the coarse model and on our judgement regarding the relative accuracy of the Monte Carlo methods for both coarse and accurate evaluations. We will describe these ideas in more detail in the context of the example in section 4.6.3.

4.6.3 Emulating $A(\theta^1, z^1)$

Beginning with S1, we must emulate $A(\theta^1, z^1)$ as a function of θ_{t_0}, θ_{t_1}, and z_{t_1}. Figure 4.17 shows the coarse evaluations that we will use to fit the coarse emulator. Although we see that the main effects seem to be roughly quadratic in z_{t_1}, we expect interactions between z_{t_1} and the decisions θ_{t_0} and θ_{t_1} to be important. We also expect individual effects from each of the decisions, reflecting our preference for abating as little as possible if it will not affect the temperature.

By fitting a saturated linear model of order two in θ_{t_0}, θ_{t_1}, and z_{t_1}, we achieved an adjusted R^2 of 0.9982 on 991 degrees of freedom. This might be an indicator of an excellent fit. Upon looking at figure 4.17, we notice that there is a very clear and dominating signal in z_{t_1}. This means that the large value of R^2 is an artifact of the fact that most of the variation in the data is due to this quadratic behaviour in z_{t_1}. This itself is an artifact of the large values of z_{t_1} that we include in our designs only because they arise occasionally when sampling from the lognormal distributions we shall use at step S4. We must be careful to ensure that, for the majority of values of z_{t_1} that we shall sample at step S4, our fit is still reasonable and that there is no hidden behaviour that is masked by the large signal in z_{t_1}. In order to explore this, we fit the same models to the subset of the data with $z_{t_1} < 3$. We found that the key features of our fits were the same in both cases, giving us confidence that our fits do not miss important information as a result of fitting on a large range of z_{t_1}. We report the details of each of these fits in appendix E.2.

On examining the residual plots in figure (4.18), we can see that the usual regression assumptions, particularly homoscedasticity in the residuals, are not valid.

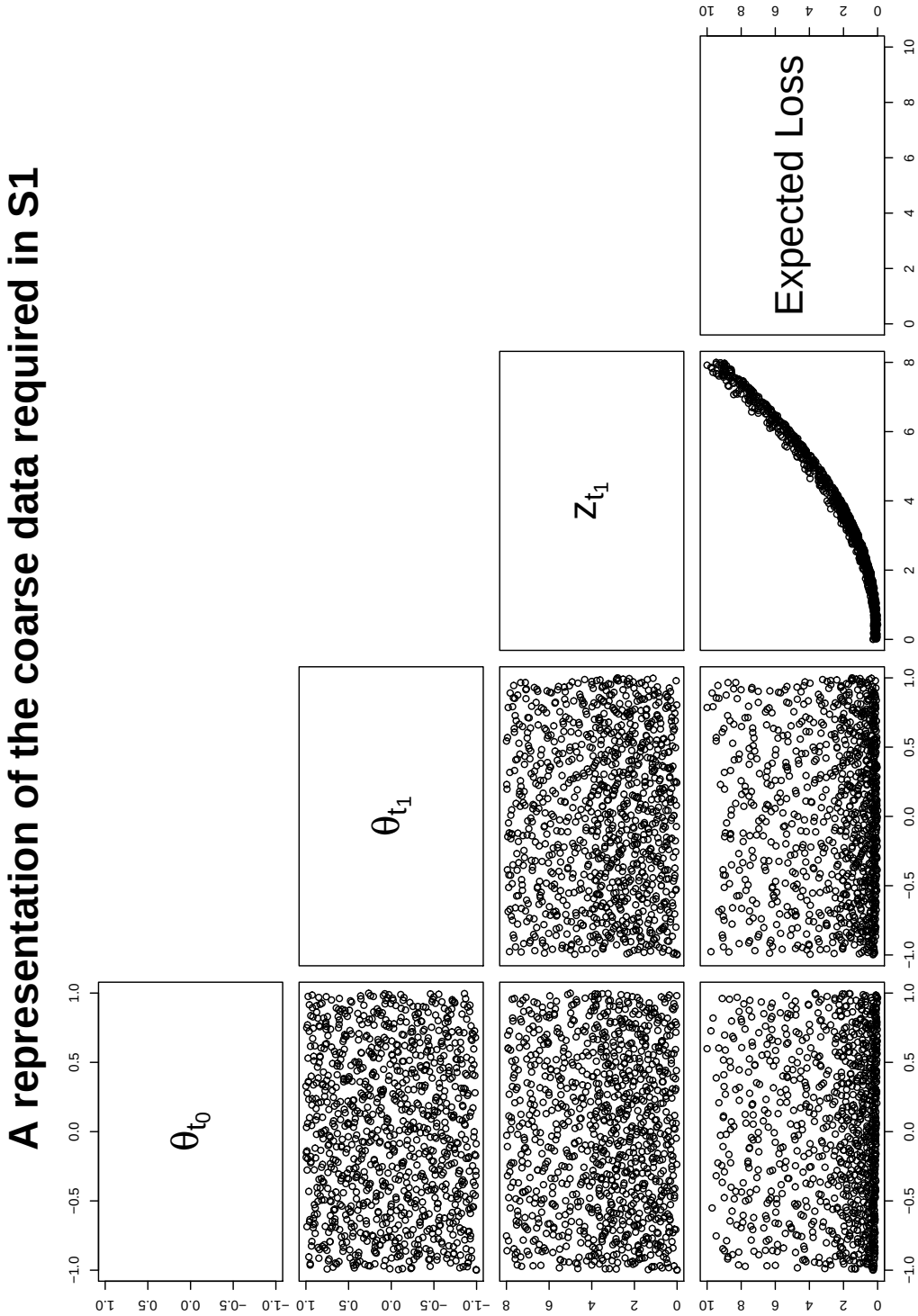

Figure 4.17: Scatter plots of our coarse evaluations of $A(\theta^1, z^1)$.

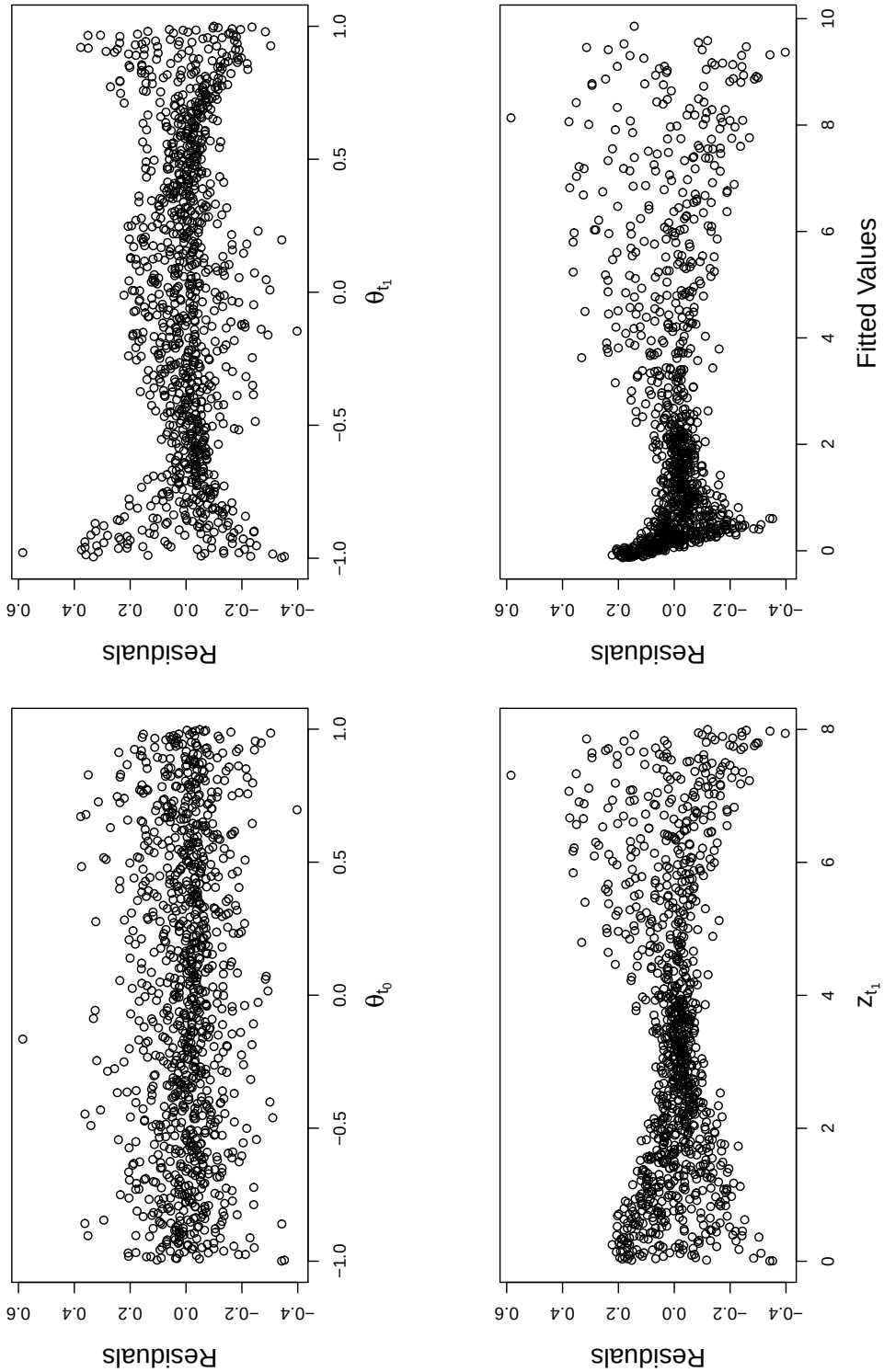

Figure 4.18: Residual plots from fitting a saturated linear model of order two to the coarse data for $A(\theta^1, z^1)$.

Although we might often be very happy to have captured over 99% of the variation in the data, as we have noted, this large R^2 is an artifact of the strong signal in z_{t_1} and we wish to ensure that we capture any smaller signals that may be a feature of the data, particularly for low values of z_{t_1}, as these will represent the majority of values of z_{t_1} that are sampled at step S4. The residuals appear to suggest a signal in z_{t_1} and θ_{t_1} in particular that our model is not detecting. As we have a large number of degrees of freedom, we take the view that it is worth adding cubic effects to our model in order to reduce the importance of the correlated residual process we are to model.

We fit a saturated linear model of order three and plot the residuals in figure 4.19. By observing these plots we can see that we have captured most of the available signal, although we still have a little remaining hetroscedasticity in z_{t_1} that will have to be modelled by our correlated process $\varepsilon^c(\theta_{t_0}, \theta_{t_1}, z_{t_1})$. We fix the coefficients β at the estimates given from the regression. To capture the hetroscedasticity seen in the residual plot for z_{t_1}, we measure the residual variance in each of the subsets $z_{t_1} \in [0, 2]$, $z_{t_1} \in [2, 4]$, $z_{t_1} \in [4, 6]$, $z_{t_1} \in [6, 7]$, and $z_{t_1} \in [7, 8]$. Our intention is to let $\sigma^2_{\varepsilon^c}$ be a function of z_{t_1}, calculated by interpolating these variances once we have removed $\sigma^2_{\delta^c}$.

We note that the residual variance is very small relative to the size of the losses we are interested in. This means that we do not need to worry too much about very careful modelling of the residual. Therefore, we choose the value of σ_{δ^c} based on the amount of residual variation that looks like white noise in figure 4.19 and based on the amount of variation we feel is due to numerical error. For this emulator we chose $\sigma^2_{\delta^c} = 5 \times 10^{-4}$, because we felt that the majority of the residual variation represented the numerical error we desire to smooth out. This left $\sigma^2_{\varepsilon^c}(z_{t_1}) \in [1 \times 10^{-4}, 3 \times 10^{-3}]$. The inverse squares of the correlation lengths, as given by our heuristic, were $(1.6, 1.6, 0.1)$.

Our emulator for the accurate calculation and, therefore, of $A(\theta^1, z^1)$, is given by equation (3.12). In order to obtain this emulator through adjusting by the difference between coarse and accurate evaluations, we must specify $E[\rho]$, $Var[\rho]$, $E[\gamma]$, $Var[\gamma]$, $\sigma^2_{\varepsilon^a}$, and $\sigma^2_{\delta^a}$. We begin by calculating the predicted values for the

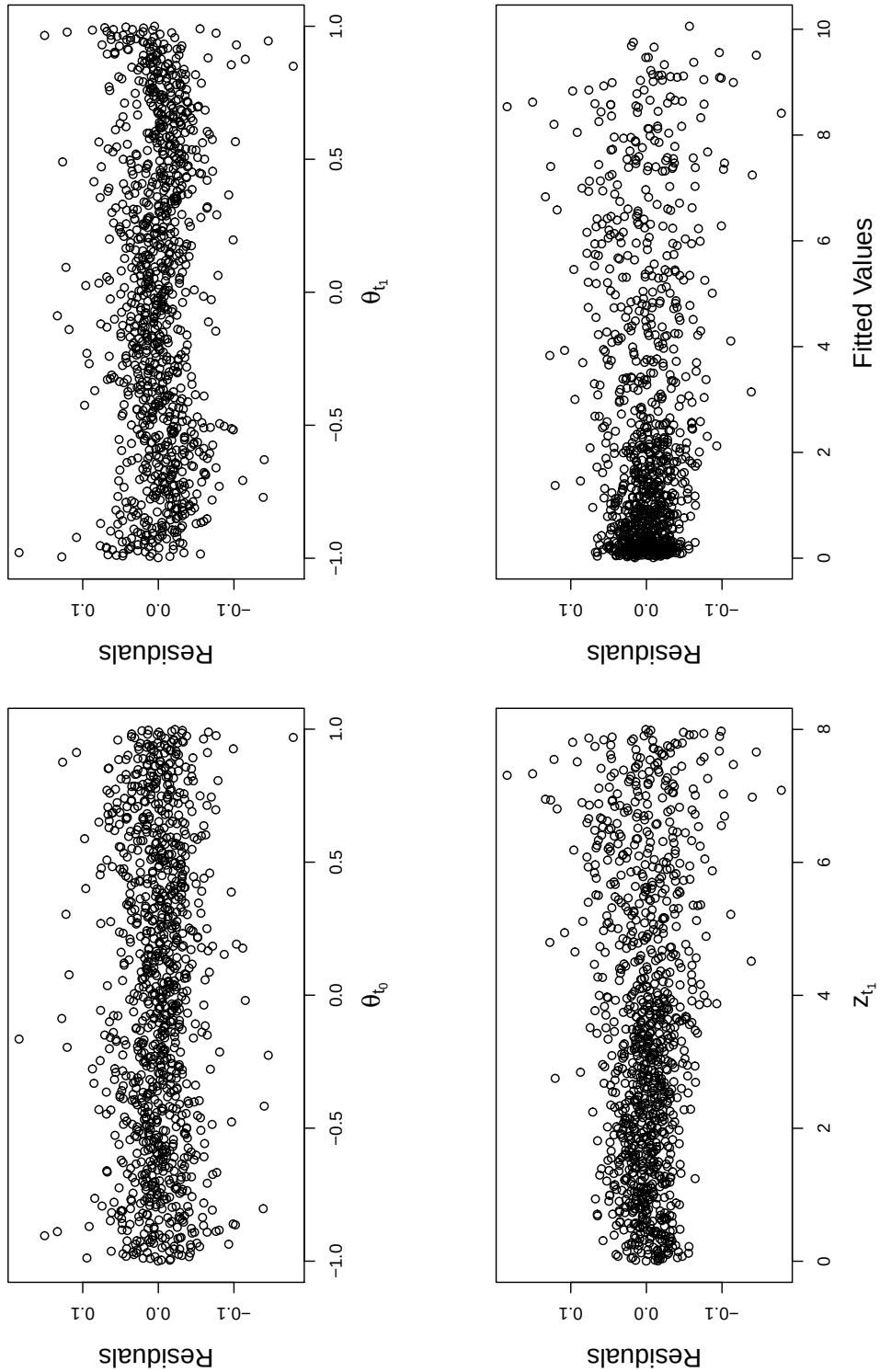

Figure 4.19: Residual plots from fitting a saturated linear model of order three to the coarse data for $A(\theta^1, z^1)$.

accurate design based on the coarse regression surface and examining the difference between these and the accurate evaluations. By measuring the variance in each of the subsets examined for the coarse, we can specify the variance of the correlated residual on the accurate given $\sigma^2_{\delta^a}$, in the same way as we did for the coarse. We base our choice of $\sigma^2_{\delta^a}$ solely on our choice of $\sigma^2_{\delta^c}$ and on the following heuristic. We know that the accuracy of a Monte Carlo integration improves with order $\sqrt{n}$ as the size of the samples, n, increases. Our accurate evaluations are based on 100 times more samples than our coarse, so we fix $\sigma^2_{\delta^a}$ by dividing $\sigma^2_{\delta^c}$ by 10, giving $\sigma^2_{\delta^a} = 5 \times 10^{-5}$. This gave $\sigma^2_{\varepsilon^a} \in [2.8 \times 10^{-4}, 2.7 \times 10^{-3}]$.

We choose $E[\rho] = E[\gamma] = 1$ and gave each quantity a variance of 0.01. We then adjust the prior emulator for the accurate by the difference between our accurate evaluations and coarse evaluations made at the same locations, using the Bayes Linear update equations and equation (3.15). The R code used to perform this adjustment is given in appendix D.2.2. The result is our emulator for $A(\theta^1, z^1)$, which we evaluate in R for any θ^1 and z_{t_1} via the function *AFine* in appendix D.2.4. The adjusted expectations of ρ and γ were 1.00162 and 1.053047 respectively. We calculated the Bayes Linear diagnostic called the *adjustment discrepancy* for both of these quantities.

If we have beliefs about quantity B and observe data D then the adjustment discrepancy is defined to be

$$Dis_D(B) = \frac{|E_D[B] - E[B]|^2}{RVar_D[B]}. \tag{4.8}$$

The adjustment discrepancies we observed for ρ and γ were 0.000263 and 0.4095. Both of these values do not flag up any serious concerns, although the very small value for the discrepancy of ρ may indicate that we put too much prior variance on ρ. However, it could also indicate that we modelled the global behaviour of the integral very well in the first place.

4.6.4 Defining time t_1 strategy, λ_{t_1}

We now follow S2 by using the function 'optimize' in R, on the expectation of our emulator for $A(\theta^1, z^1)$ to locate $\lambda_{t_1}(\theta_{t_0}, z^1)$ for any θ_{t_0}, z_{t_1}. We first calculate the

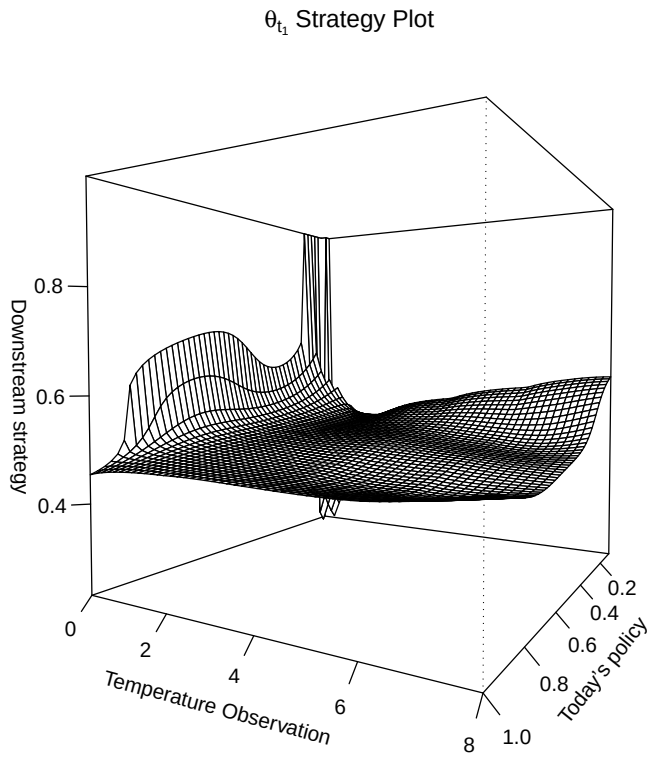

Figure 4.20: A Strategy plot for θ_{t_1}, where the surface $\lambda_{t_1}(\theta_{t_0}, \theta^1)$ is plotted as a function of θ_{t_0} and z_{t_1}.

minimum on the whole range of possible θ_{t_1} and then on half intervals of that range to insure against landing in a local optimum with our algorithm. Figure 4.20 shows the first of our strategy plots for λ_{t_1}. We discuss the information conveyed by this plot when presenting our policy support ideas in section 4.7.

4.6.5 Emulating $B^1_{\lambda^1}(\theta_{t_0}, z^1)$

We proceed to S3 and re-emulate $A(\theta^1, z^1)$ as a function of θ_{t_0} and z_{t_1} only, substituting θ_{t_1} with $\lambda_{t_1}(\theta_{t_0}, z^1)$. We obtain coarse and accurate evaluations of $B^1_{\lambda^1}(\theta_{t_0}, z^1)$ using the design mentioned in section 4.6.1. The data for the coarse is shown in figure 4.21.

We fit a saturated linear model of order two and obtain an adjusted R^2 of 0.9998. This very large value of R^2 comes from the fact that the data in figure 4.21 appear

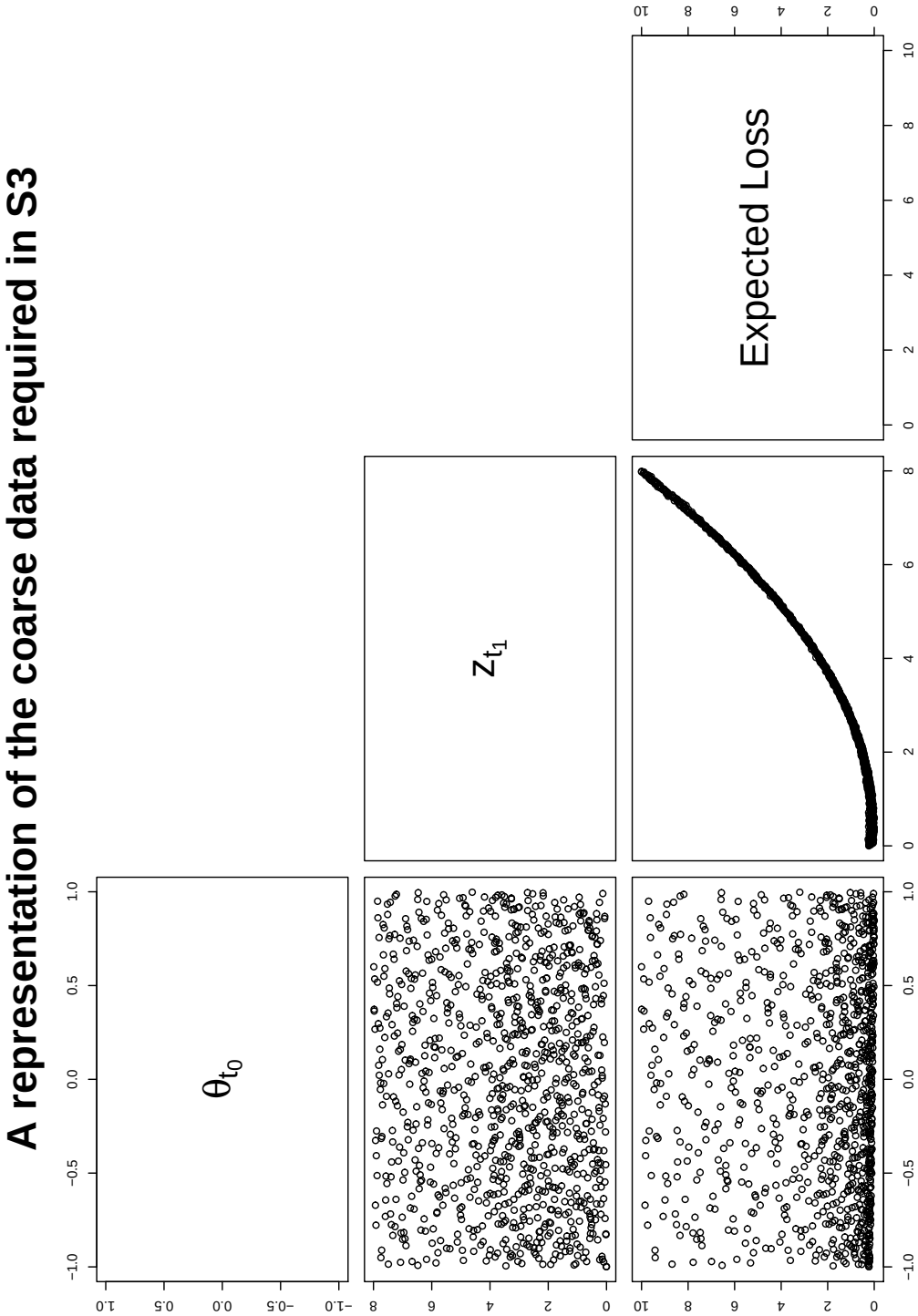

Figure 4.21: Scatter plots of our coarse evaluations of $B^1_{\lambda^1}(\theta_{t_0}, z^1)$.

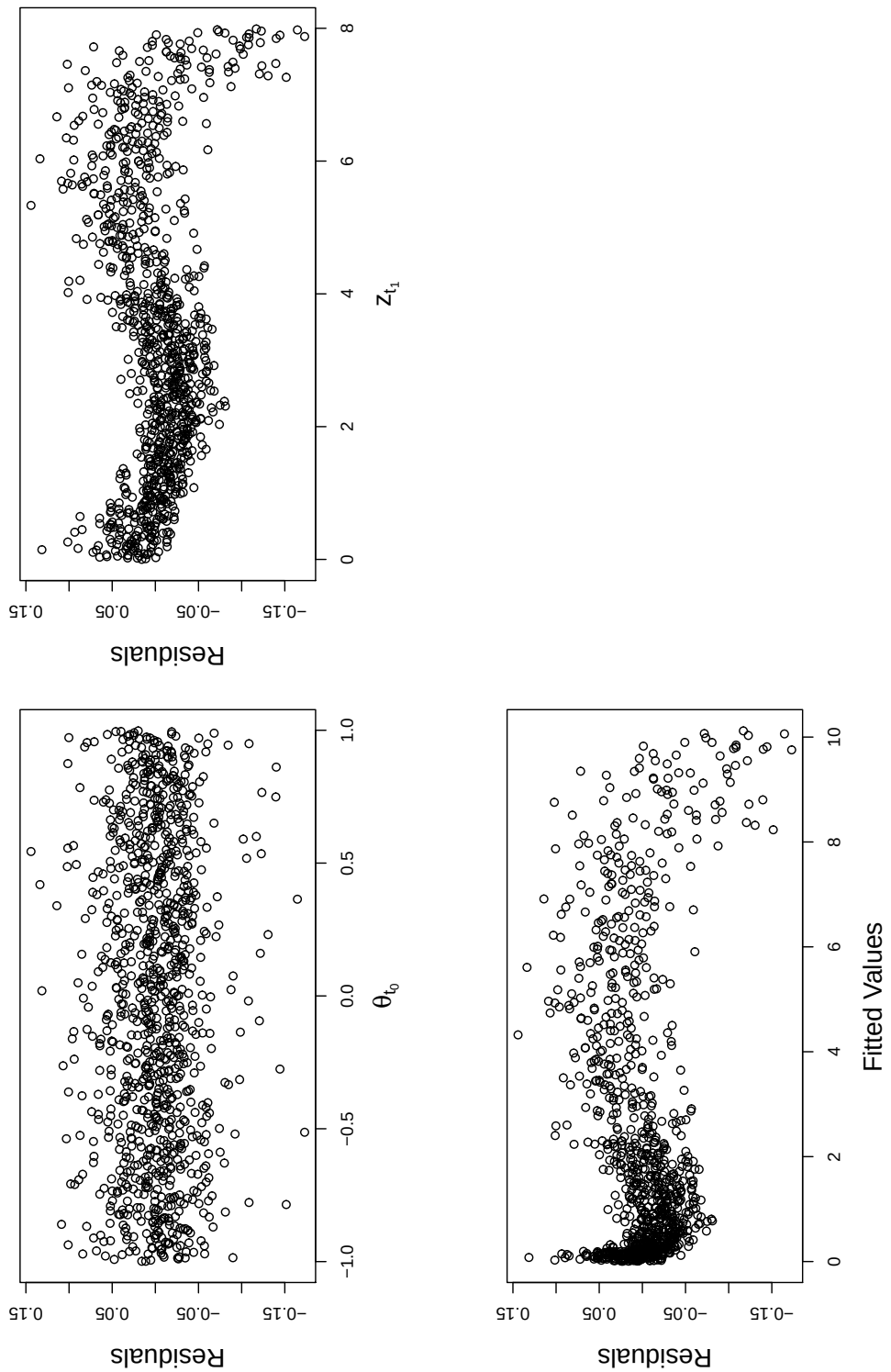

Figure 4.22: Residual plots from fitting a saturated linear model of order two to the coarse data for $B^1_{\lambda^1}(\theta_{t_0}, z^1)$.

to be a quadratic curve in z_{t_1}. We check that the global fit over the entire range of z_{t_1} does not hide a different type of signal for those values of $z_{t_1} < 3$ that will be most sampled from the log-normal distribution at step S4, by fitting the model over the subset of the data with $z_{t_1} < 3$. The summaries for both fits are in appendix E.2. The coefficients for both fits were the same sign and order of magnitude, giving us confidence that the large R^2 observed here is not misleading and that we do capture the behaviour of the data very well. The residuals for this fit are shown in figure 4.22. These residuals display some hetroscedasticity as well as suggesting that there is a higher order signal there. Although adding the order three terms seems to capture some of this signal, as shown in figure 4.23, the residual variance is only reduced very slightly. We present the summary for this cubic fit in appendix E.2 and argue there that it may not be worth adding the extra terms. We choose the quadratic model and allow the correlated residual to capture any remaining signal, as the size of this signal is very small.

We fit the new $\sigma^2_{\varepsilon^c}$ and $\sigma^2_{\varepsilon^a}$ in the same way as our previous multi-level emulation, allowing the variance to grow with z_{t_i}. We again choose $\sigma^2_{\delta^c} = 5 \times 10^{-4}$ and $\sigma^2_{\delta^a} = 5 \times 10^{-5}$. We do this based on the knowledge that our numerical integration here is of the same loss function with respect to the same types of 3-dimensional log-normal distributions in y, and using the same sample sizes for both coarse and accurate evaluations as our numerical integration used to generate evaluations of $A(\theta^1, z^1)$. We also choose the same mean and variance for ρ and γ as with our previous emulation. Our heuristic for choosing correlation length sets the inverse square correlation lengths on both coarse and accurate correlated processes to be 0.9 and 0.05625 for θ_{t_0} and z_{t_1} respectively.

We now adjust our prior model for the accurate calculation by the difference between accurate and coarse runs made at the locations of our 200 point Latin Hypercube design. The adjusted expectations of ρ and γ were 1.008151 and 0.9082235 respectively. The discrepancies, calculated as in section 4.6.3, were 0.006646 and 0.8425 for ρ and γ respectively. Again, we may consider that our prior variance for ρ was too large, or that we captured the global behaviour of the expected loss very well using the coarse evaluations.

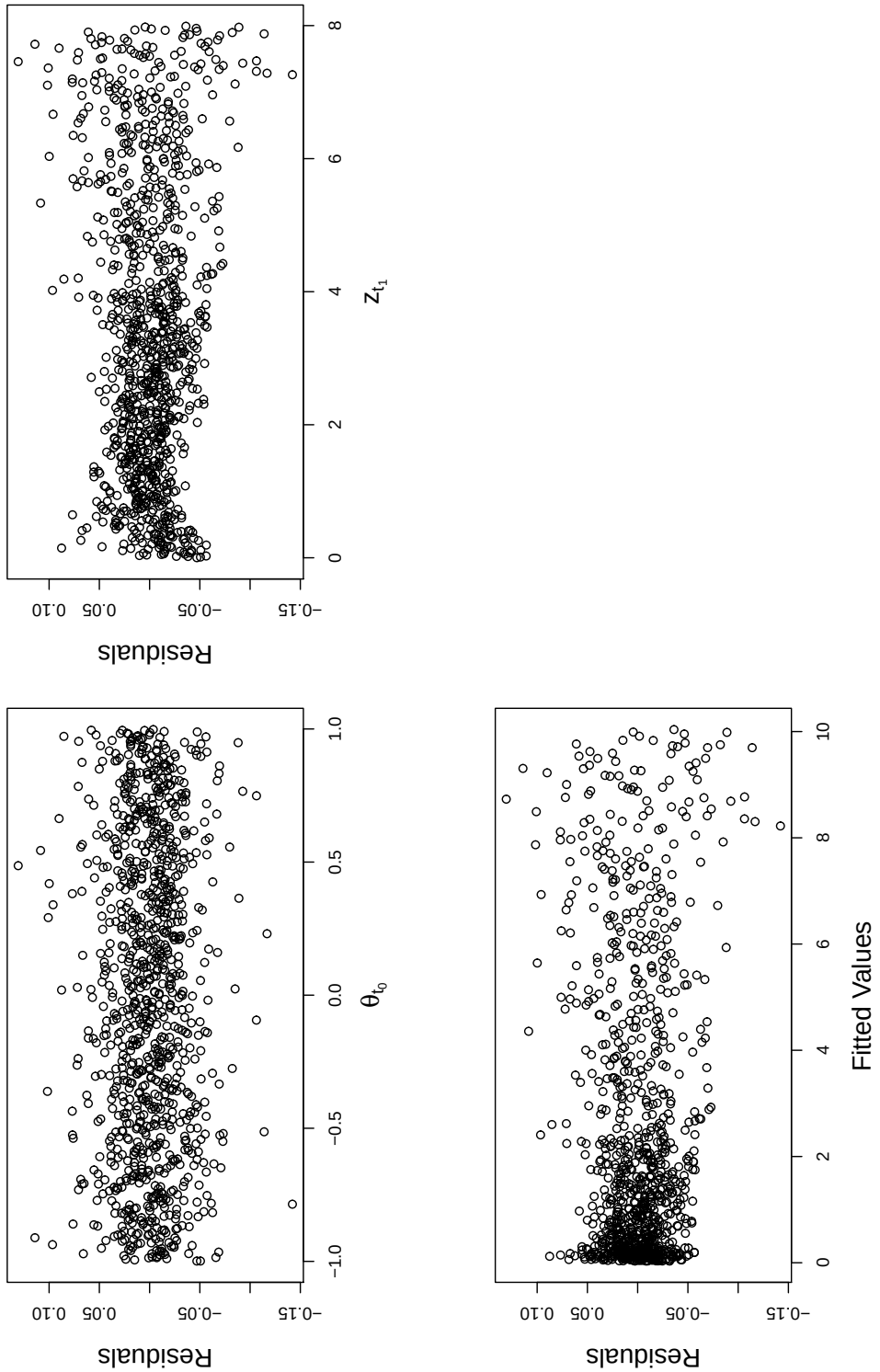

Figure 4.23: Residual plots from fitting a saturated linear model of order three to the coarse data for $B^1_{\lambda^1}(\theta_{t_0}, z^1)$.

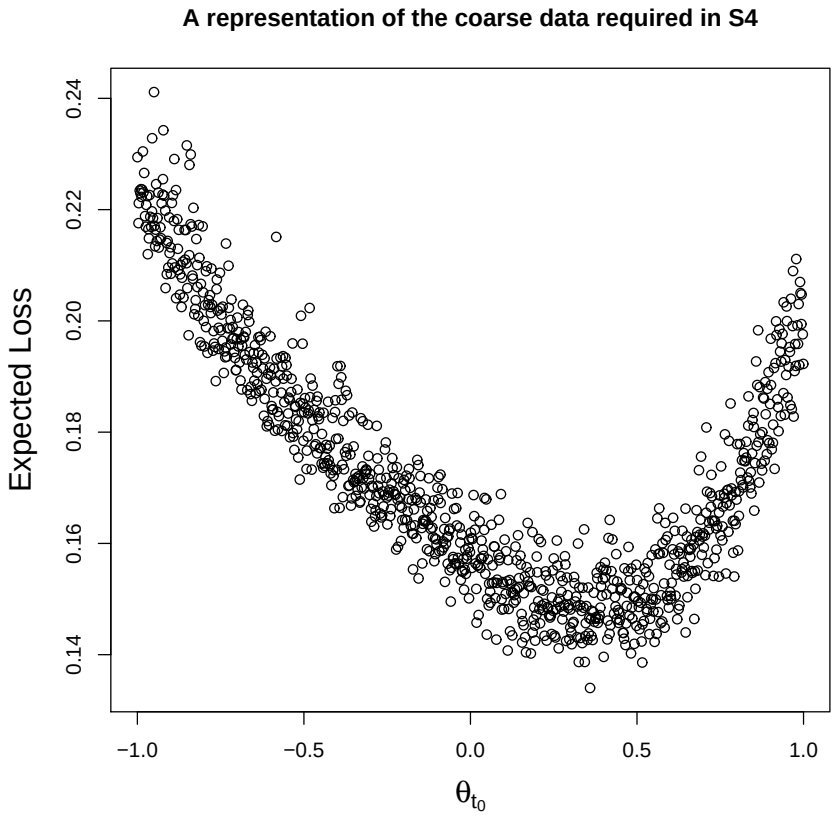

Figure 4.24: A scatter plot of our coarse evaluations of $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$.

4.6.6 Emulating $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$

At step S4 we must emulate the function

$$C^1_{\lambda^1}(\theta_{t_0}, z_{t_0}) = \int \tilde{E}\left[B^1_{\lambda^1}(\theta_{t_0}, z^1)\right] p(z_{t_1}|z_{t_0}, \theta_{t_0}) dz_{t_1} \tag{4.9}$$

which, as z_{t_0} is fixed, is a function of θ_{t_0} only. For each point in our coarse and accurate designs we obtain evaluations of (4.9) by sampling the required number of future temperatures from $p(z_{t_1}|z_{t_0}, \theta_{t_0})$ in the manner discussed in section 4.5.3 and evaluating the expectation of our emulator for $B^1_{\lambda^1}(\theta_{t_0}, z^1)$ for each sample. The mean of these evaluations is our estimate.

We present the coarse evaluations in figure 4.24. We can see from this diagram that there is a clear quadratic signal in θ_{t_0}, and that the minimum expected loss will be located somewhere between $\theta_{t_0} = 0$ and $\theta_{t_0} = 0.6$ We try fitting a quadratic polynomial in θ_{t_0} via least squares and obtained an adjusted R^2 of 0.8768.

The residuals from this fit are shown in figure

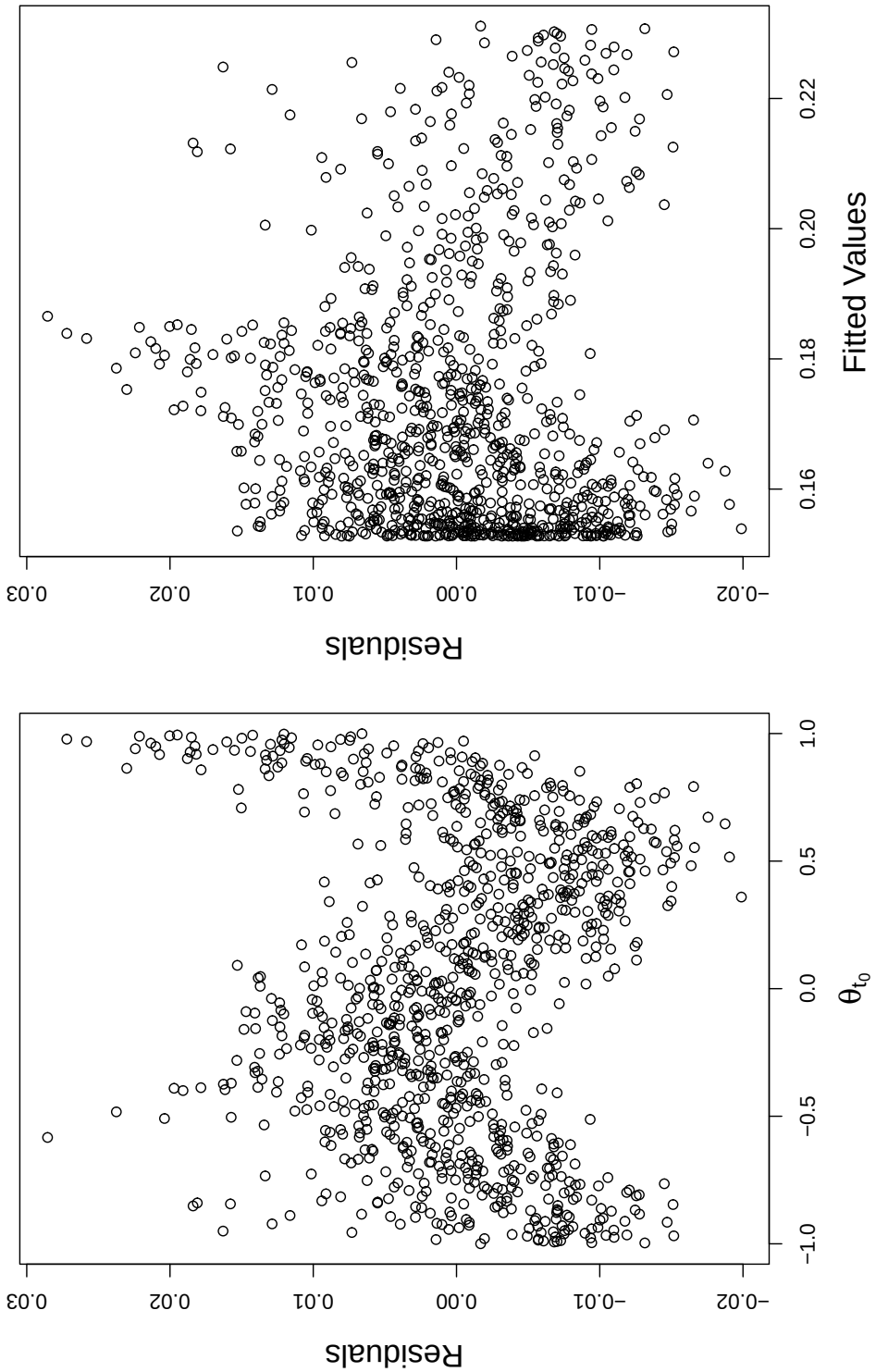

Figure 4.25: Residual plots from fitting $\theta_{t_0} + \theta_{t_0}^2$ to the coarse data for $C_{\lambda^1}^1(\theta_{t_0}, z^1)$.

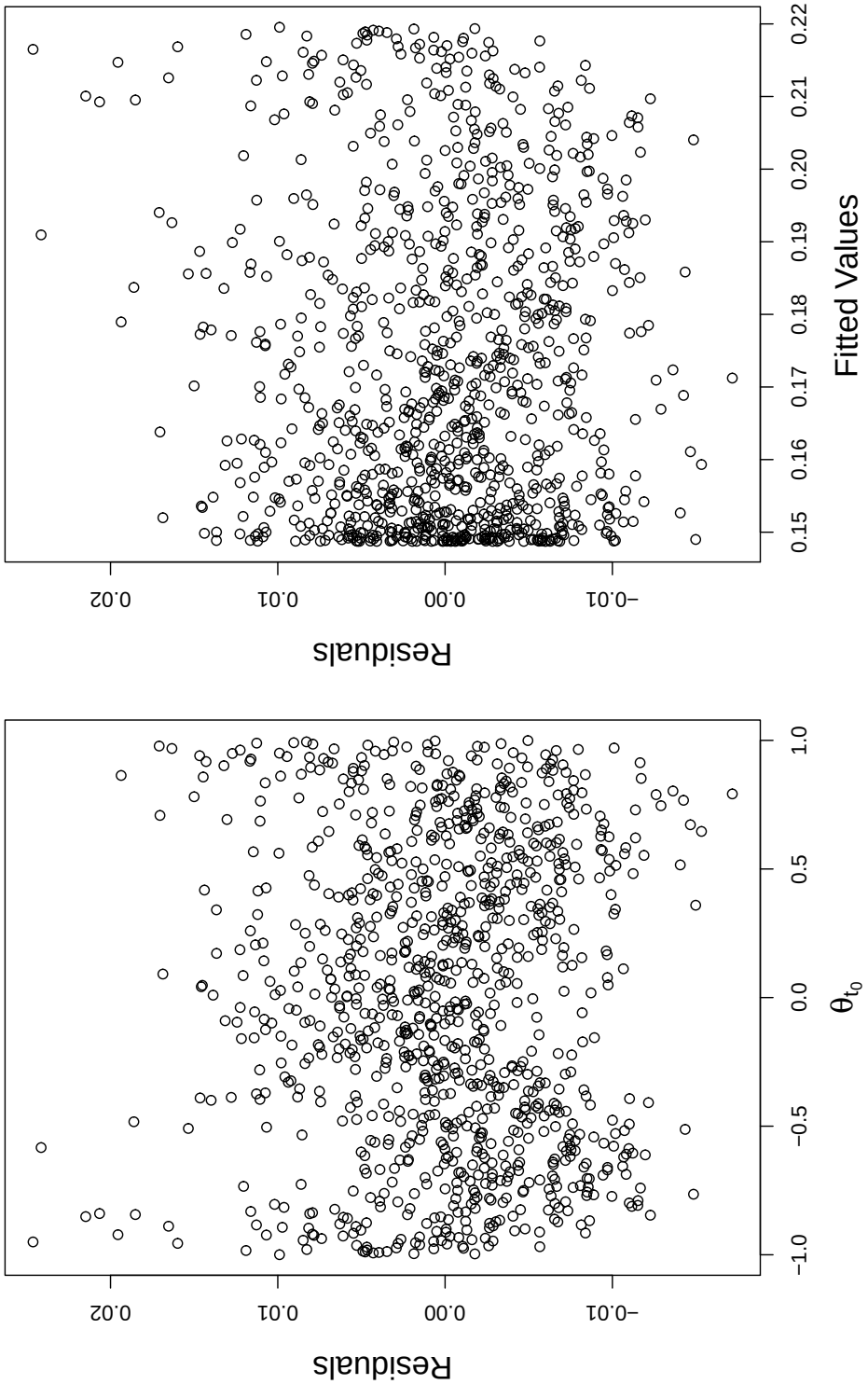

Figure 4.26: Residual plots from fitting $\theta_{t_0} + \theta_{t_0}^2 + \theta_{t_0}^3$ to the coarse data for $C_{\lambda^1}^1(\theta_{t_0}, z^1)$.

4.25. The residuals display evidence of higher order signal and, in fitting a cubic polynomial, we were able to obtain an adjusted R^2 of 0.9166. The residuals from this fit are shown in figure 4.26. They seem reasonably homoscedastic and have no obvious patterns remaining, suggesting that any residual variation may be uncorrelated white noise. The variance of the residuals for both the coarse and accurate evaluations is 4.03×10^{-5} and 8.23×10^{-6} respectively. We reflect our judgements regarding the amount of uncorrelated variation contained in the residual by choosing $\sigma^2_{\delta^c} = 4 \times 10^{-5}$ and $\sigma^2_{\delta^a} = 4 \times 10^{-6}$.

Our correlation parameter, as fixed using the same heuristic we have used throughout, is 1.6 and we fix the same prior beliefs on ρ and γ as with our previous emulators. We now adjust our prior model for the accurate calculation by the difference between accurate and coarse evaluations, as usual, to obtain our emulator for $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$. The adjusted expectation of ρ and γ was 0.99198 and 1.01723 respectively. The discrepancies for these quantities were 0.00645 and 0.61855. The expectation of this emulator provides us with our visualization of the loss surface, shown in figure 4.27. This image, along with the strategy plot in figure 4.20, represents our first decision support tool. Moving to S5 in the algorithm, we see that $m = 1$ and so we proceed to S11 and stop. Having completed the Sequential Emulation algorithm, we may now focus on providing decision support.

4.7 Policy support

In this section we present some of the methods of policy support we can offer using the emulators obtained during a Sequential Emulation and some of the methods discussed in section 3.7.

4.7.1 Forward sampling

Having fixed all downstream strategies, $\lambda_{t_1}, \ldots, \lambda_{t_m}$ (in this case only λ_{t_1} was required) we may forward sample in the way described in section 3.7.3 to obtain the risk profile for a given θ_{t_0}. In order to simulate the future behaviour of the temperature anomaly in response to a particular policy θ_{t_0} and downstream strategy λ_{t_1},

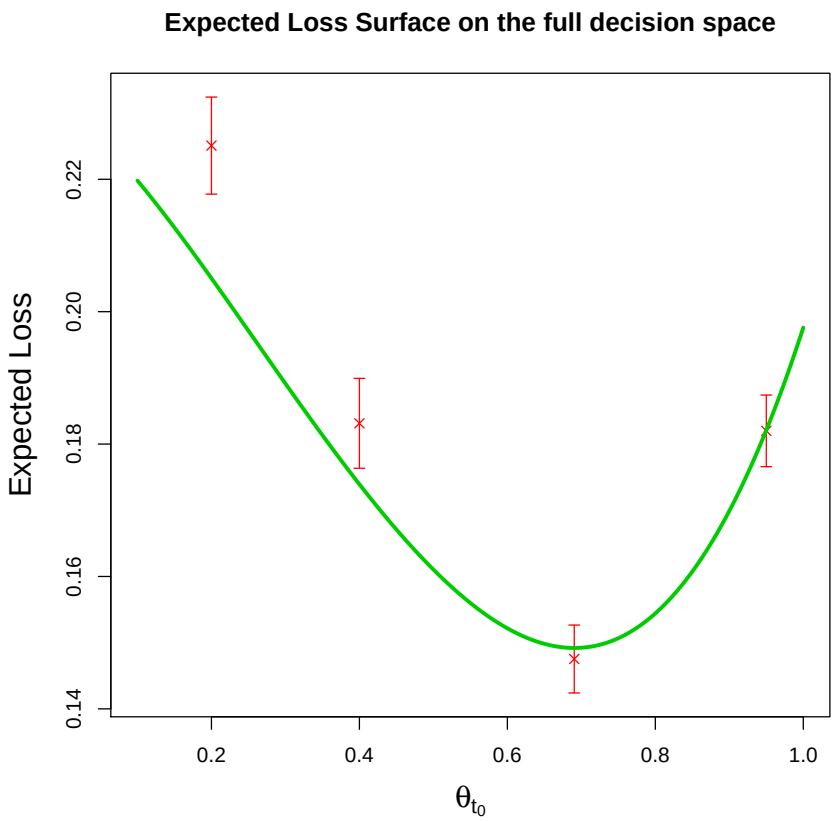

Figure 4.27: A plot of $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ with error bars obtained through the forward sampling methods described in section 4.7.1.

we take the following steps. We first sample a value from $p(z_{t_1}|z_{t_0}, \theta_{t_0})$ and call this sample $\hat{z}_{t_1}$. Using our downstream strategy λ_{t_1} we compute

$$\hat{\lambda}_{t_1} = \lambda_{t_1}(\theta_{t_0}, (z_{t_0}, \hat{z}_{t_1})).$$

We then sample a value $\hat{y}$ from $p(y|\hat{z}_{t_1}, \theta_{t_0}, \hat{\lambda}^1)$, characterized using a Bayes Linear forecast based on the sampled observations and our policy strategy, and compute $L(\hat{y}, (\theta_{t_0}, \hat{\lambda}^1))$ using DICE. This gives us one sample loss value for a given θ_{t_0}. By repeating these steps a large number of times, we obtain a picture of the loss distribution under our downstream intervention strategy for a given θ_{t_0} and we may use the results as a guide to how well our Sequential Emulation captures our expected loss for the strategy and to what extent that strategy might be improved.

The points and error bars in figure 4.27 show the simplest use of this sampling. For given θ_{t_0} we can obtain the risk profile and use the sample mean and its standard

error to plot actual, achievable, expected losses under strategy λ_{t_1}. Using these, we may assess how well our Sequential Emulation captures the loss surface under λ_{t_1}. For each of the values $\theta_{t_0} = 0.2$, 0.4, 0.703 and 0.95 we conducted a sampling experiment based on 60000 simulations. The red crosses and error bars on figure 4.27 were then obtained by placing a cross at the mean of each sample, and putting the error bars at a distance of $\pm$ 2 standard errors of the sample mean. We can see from figure 4.27 that our Sequential Emulation has captured expected loss under λ_{t_1} near the minimum of the curve well, but under-estimates expected loss for the 'sub-optimal' decisions that we have explored. We could tweak our emulators by obtaining more evaluations in the areas in which we are under-performing if we so wished. However, our intention is to prune the tree and, as will be argued in the next section, our Sequential Emulation is adequate for that purpose.

One of the principle concerns when performing a Sequential Emulation must be with how close our strategies $\lambda_{t_1}, \ldots, \lambda_{t_m}$ (in our case, just λ_{t_1}) are to optimal. Put another way, we would like to know by how much our intervention strategies might be improved. Forward sampling can give us insight into these problems. For a chosen θ_{t_0}, we may sample a z_{t_1} as usual and compute $\lambda_{t_1}(\theta_{t_0}, z^1)$. We can then sample $y(\theta_{t_0}, \lambda_{t_1})$ and $L(y, (\theta_{t_0}, \lambda_{t_1}))$ as normal, but we may also take a handful of alternative downstream strategies forward and sample losses for these. By taking a large sample of losses for each alternative strategy, we may compare performance of λ_{t_1} to alternative schemes. This provides a simple diagnostic for our emulator for $A(\theta^1, z^1)$, but, more importantly, will be a useful tool to aid pruning.

4.7.2 Pruning

Our goal in pruning, as discussed in section 3.7, is to remove areas of the decision space that we do not believe contain optimal decisions. We may then refocus the analysis and provide improved downstream intervention strategies and more insight into our expected loss surface for policy today.

We may be tempted, having observed figure 4.27, to remove all θ_{t_0} that result in an expected upper bound larger than, say, 0.18. This would mean reducing the range of θ_{t_0} under consideration to roughly $[0.4, 0.95]$. However, we are not certain whether

alternative strategies may result in some θ_{t_0} outside of that range becoming optimal. Another problem with this is that under λ_{t_1} our curve is merely an emulator for an upper bound on expected loss. The real upper bound may be lower in areas of the decision space we are considering throwing away than it is at decisions that appear to be better according to our emulator. We may be comforted by the location of the actual expected losses we have plotted.

Prior to performing this detailed analysis, we explored the behaviour of the loss function for many different parametrizations and for the types of temperature values we would expect to come from our forecasting calculations. We did this primarily in order to choose values for the loss function parameters that might provide an interesting story. However, this initial exploration, combined with a number of early tests of the Sequential Emulation methodology lead us to be reasonably confident that our expected loss surface is a smooth convex function. We, therefore, do not expect lower expected losses under λ_{t_1} for θ_{t_0} outside, say, $[0.4, 0.95]$ than for θ_{t_0} inside that range. We cannot, however, ignore the potential for improved downstream strategies to move the minimum of our function to significantly different areas of the decision space. We therefore turn to strategy plots and consider learning about the possible impact of alternative strategies and addressing the problem of pruning the space of θ_{t_1}.

We first look at strategy plots for $\theta_{t_0} = arg\min_{\theta_{t_0}}\{\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]\}$. Figure 4.28 is such a plot representing expected losses for one sampled z_{t_1} under this policy, with the corresponding λ_{t_1} and a number of alternative strategies explored. Each expected loss and associated standard error is based on $10,000$ samples of y given θ_{t_0}, the sampled z_{t_1}, and the appropriate downstream strategy. The point highlighted in blue corresponds to expected loss under λ_{t_1}. The strategy plot in figure 4.28 shows that λ_{t_1} performed better than the other strategies tested for the sampled z_{t_1}. What we also observe from this plot is that we have higher expected losses for $\theta_{t_1} \notin [-0.5, 0.5]$ than for those strategies inside that interval. Note that on these plots θ_{t_1} has been mapped to $[-1, 1]$ to make Sequential Emulation easier. Final reporting of plausible strategies would be done with θ_{t_1} mapped back to $[0.1, 1]$. If we were to see this pattern for a large number of z_{t_1} and for multiple θ_{t_0}, we

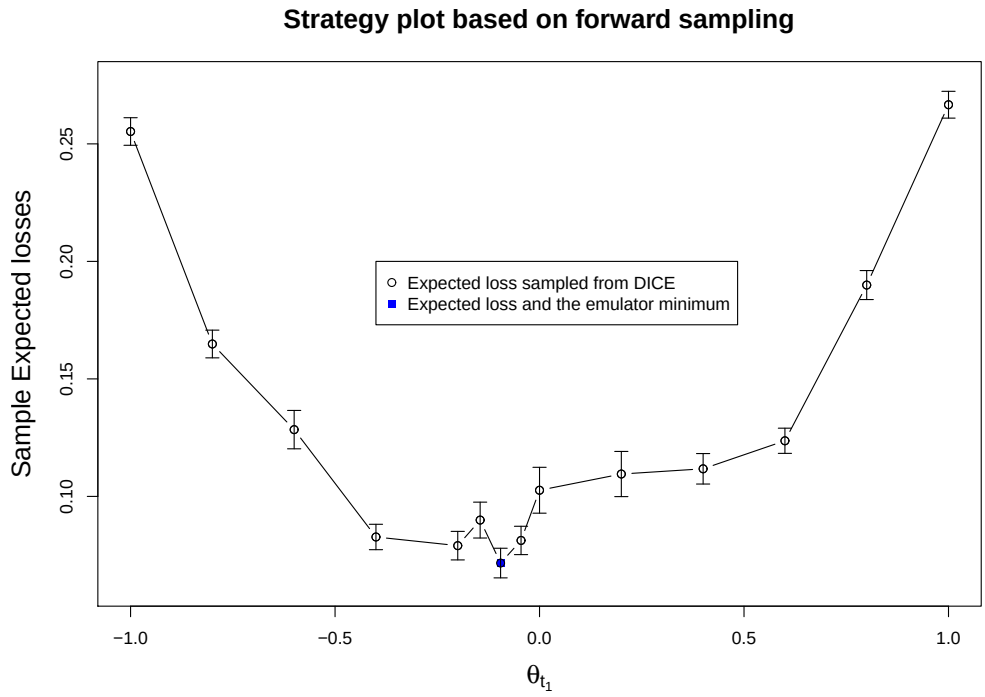

Figure 4.28: Strategy plot for $\theta_{t_0} = arg\min_{\theta_{t_0}}\{\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]\}$.

could refocus our Sequential Emulation by pruning the tree so that $\theta_{t_1} \in [-0.5, 0.5]$. Figure 4.29 and figures E.1, E.2, E.3 presented in appendix E.3 show a number of these strategy plots for different sampled z_{t_1} and for four different choices of θ_{t_0}. The red dotted lines indicate where we intend to prune the space of possible θ_{t_1} for given θ_{t_0}. What we are hoping to see in each of these plots is that no strategy sampled outside the interval defined by these dotted lines does better than all of those sampled within the interval. If we do find such strategies, we must consider being more conservative in our efforts to prune the space and perhaps obtain further samples.

Figure 4.29 seems to indicate that as θ_{t_0} has decreased (corresponding to more abatement today), optimal θ_{t_1} seems to have increased. Looking at figure 4.20, we may expect this, however, the panels suggest that if choosing to abate significantly today, there may be optimal strategies outside of the areas of the decision space we intend to keep (although strictly we do not observe any). The strategies, θ_{t_1}, that we may prune from our decision tree then, are greatly influenced by those areas of the space of possible θ_{t_0} that we allow.

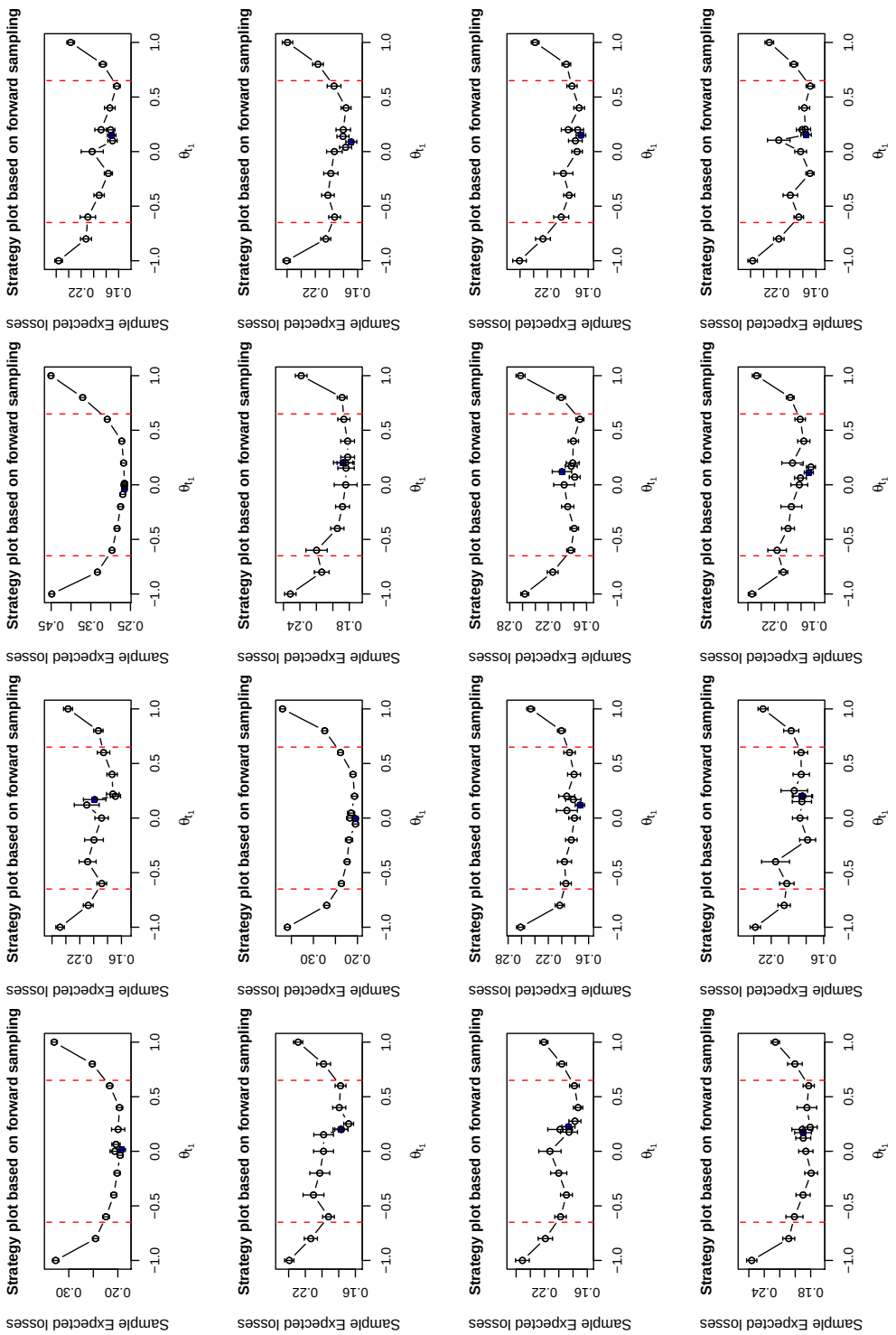

Figure 4.29: Panel of strategy plots for $\theta_{t_0} = 0.2$. Each panel corresponds to one sampled z_{t_1}. Red dotted lines on each panel indicate $\theta_{t_1} = -0.65$ and $\theta_{t_1} = 0.65$.

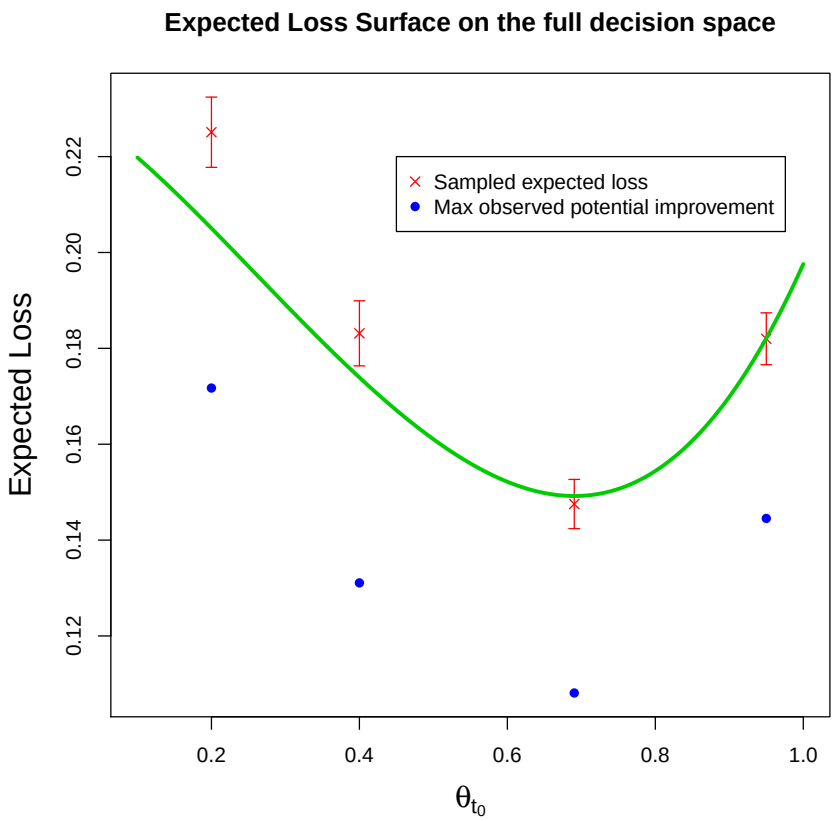

Figure 4.30: The image of $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ with the blue points indicating where our expected loss would be if we chose an intervention strategy that obtained the highest maximum observed potential improvement observed from all of our samples.

Forward sampling under alternative strategies can also give us an idea of how much improvement we might make with a 'better' downstream strategy than λ_{t_1} in place. For a chosen θ_{t_0} and sampled z_{t_1}, we define the *maximum observed potential improvement* to be the maximum difference between the top of our error bar on the expected loss under λ_{t_1} and the bottom error bar for any other sampled strategy. The maximum of this quantity over all sampled z_{t_1} gives an order of magnitude idea for how much our expected loss could be reduced by finding the true optimal downstream intervention strategy for a given θ_{t_0}.

We can use this rough statistic to plot points on the diagram of our emulated upper bound for the loss surface; this will give an idea as to how much improvement to our current expected loss we might make by following better strategies. We plot

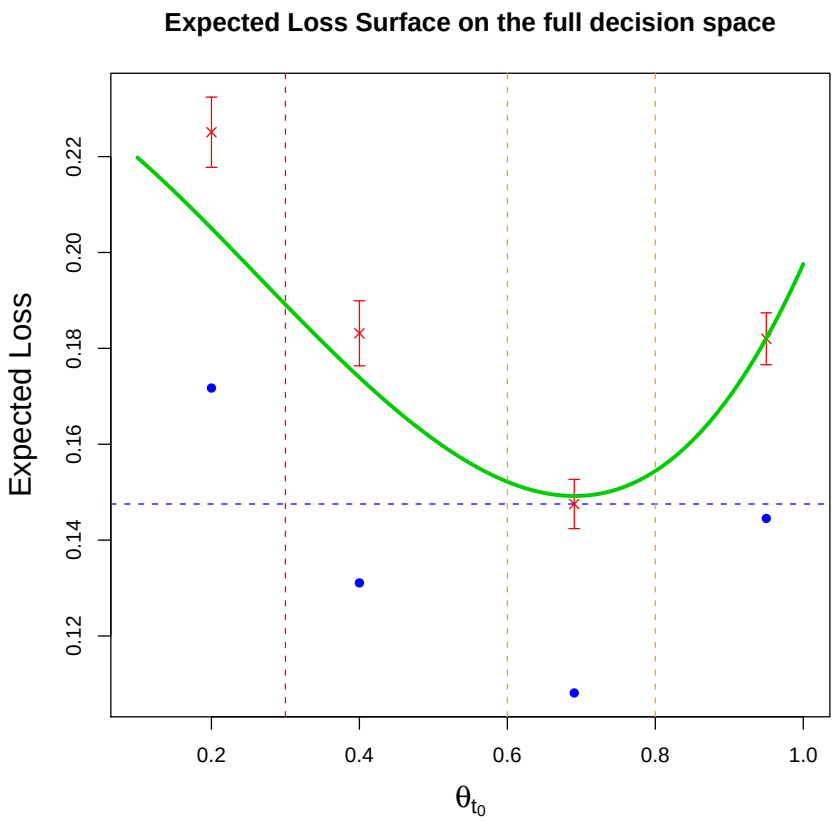

Figure 4.31: The image of $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ with the blue points indicating where our expected loss would be if we chose an intervention strategy that obtained the highest maximum observed potential improvement observed from all of our samples. Yellow dotted lines indicate the area we intend to focus our sampling, whilst the other lines are aids for pruning.

these points for our example in figure 4.30. Note that the points in red are expected losses that are achievable under λ_{t_1}. The blue points represent an order of magnitude idea for how much expected loss at sampled points may decrease if a better strategy than λ_{t_1} could be found. We use this plot to aid pruning. As we still believe our loss surface to be a parabola and because we can achieve expected loss under λ_{t_1} that is lower than our beliefs about potential improvement at $\theta_{t_0} = 0.2$, it is reasonable to reject any $\theta_{t_0} \leq 0.2$ as potential candidates for optimal policies.

In fact we go further than this. Figure 4.31 shows a vertical line at $\theta_{t_0} = 0.3$. Our pruning decisions are based on a notion that under optimal time t_1 strategy

our loss surface would be a parabola going through the blue points. In truth, this is unlikely as the maximum of the maximum observed potential improvement is designed to overestimate potential improvement and is only an order of magnitude guide. However, this does provide us with a decision support tool that advocates pruning conservatively. We can achieve the expected loss indicated by the horizontal blue line by following λ_{t_1} and by choosing $\theta_{t_0} = arg\min_{\theta_{t_0}}\{\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]\}$. By imagining a line connecting the blue dots at $\theta_{t_0} = 0.2$ and $\theta_{t_0} = 0.4$, we can imagine that this line intersects with our minimum observed expected loss under λ_{t_1} at $\theta_{t_0} = 0.3$. We therefore prune the decision tree by restricting $\theta_{t_0} \in [0.3, 1]$. We also restrict $\theta_{t_1} \in [0.2575, 0.8425]$ ($[-0.65, 0.65]$ if mapped to $[-1, 1]$), as having removed the very large abatement scenarios from our set of possible actions today, our strategy plots give us confidence that we are not removing any potentially optimal strategy options.

Note that our pruning in this example has simply reduced the size of of the decision space by removing entire regions of the domain of each of the individual decisions. There is no reason why this should happen in general. Pruning may often involve removing areas of the decision space that leave a complicated subspace. For example, we might still see all of the possible values of each individual decision in a given pruned subspace, but certain combinations of decisions are removed from our feasible set of decisions.

The actions we have taken and the tools used to inform these actions are just some of the things we might have looked at in order to prune the decision tree. Having performed a Sequential Emulation of the decision tree, forward sampling and strategy plots can be used to gain valuable insight into the quality of our emulation, the performance of our intervention strategies, and the nature of the expected loss surface. The way in which one may use these tools to prune the tree or provide any other type of decision support is not restricted to the examples we have given or, indeed, on any of those ideas mentioned in section 3.7, but is just an illustration of the potential uses of Sequential Emulation as a decision support tool.

Having pruned the decision space, we perform another Sequential Emulation of the decision tree. Ideally, as part of a case study, we would like to obtain further runs from C-GOLDSTEIN and re-emulate it on the reduced space first. However,

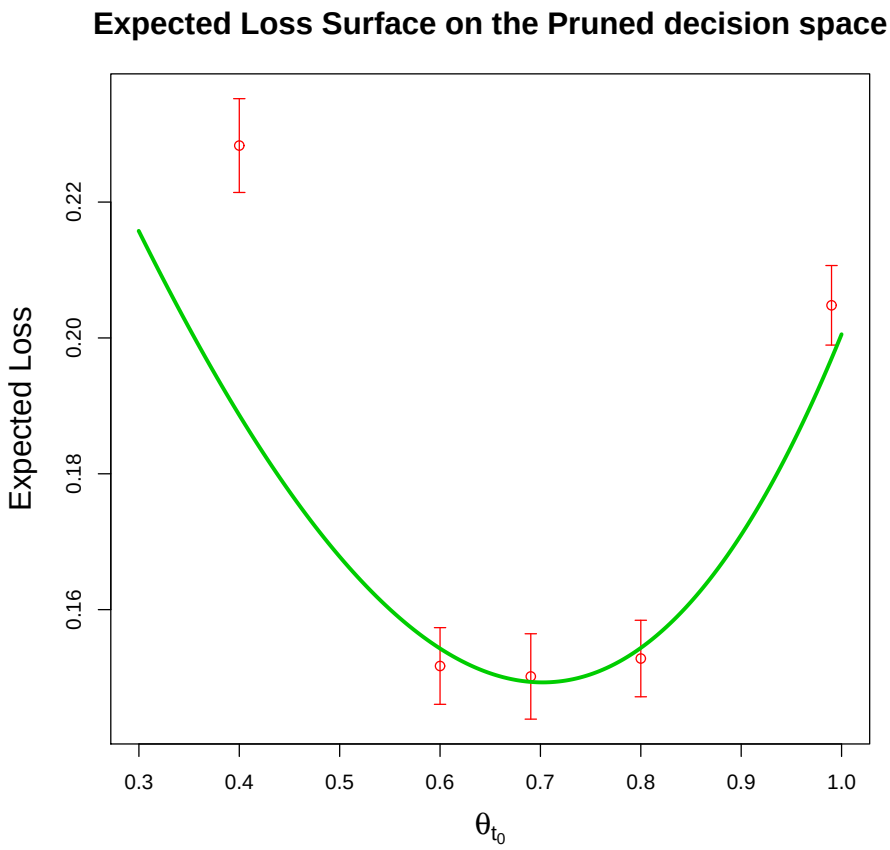

Figure 4.32: A plot of $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ for the Sequential Emulation performed on the pruned decision tree.

without direct access to the model this was infeasible. We therefore return directly to S1 and proceed as before using new designs on the pruned decision space. We add a collection of extra points to our design within the region where we suspect the optimal policy to lie. This region is highlighted between the two dashed yellow lines in figure 4.31. The resulting emulator for the expected upper bound on the loss surface is shown in figure 4.32. We present selected details from this Sequential Emulation in appendix E.3.2. We note that our emulator for the expected loss surface on the pruned decision space is very similar to our original expected loss surface and both have roughly the same minimum. This could be an indication that our original Sequential Emulation was good, particularly around the minimum. In a real-world analysis we may well prune again, refocussing our emulators until we are happy to make policy.

4.7.3 Risk profiling

Pruning is only one important avenue of decision support we might offer. Another important one is showing the risk profile for any policies of interest. If the output from DICE really was utility and, moreover, if we could solve the decision problem exactly, there would be no need to look at the risk profile as all risk issues would have been fully accounted for. However, as we have discussed, the output of DICE is not really a utility and we are emulating it anyway. Therefore, having used our methods to identify decisions with low expected loss, it is important to look at the distribution of the risk associated with any strategy we are considering pursuing. We obtain the risk profile, as discussed in section 3.7.3, by forward sampling in the way described at the beginning of this section. Having done this we may simply draw the risk profile as a histogram.

We give an example of the risk profile for $\theta_{t_0} = arg\min_{\theta_{t_0}}\{\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]\}$ under our pre-pruned strategy λ_{t_1} in figure 4.33. The figure shows the loss distribution with various ranges in focus. A noteworthy feature of this risk profile is that the expected loss, which is 0.149, is within the tail of the distribution and that most of the mass is located at more favourable outcomes. The length of the tail may also be particularly worrying or surprising to a decision maker. The shape of our risk profile is probably most heavily influenced by our choice of the log-normal distribution for future temperatures and observations. This may not always be the case and could be the result of the relative monotonicity of λ_{t_1} in z_{t_1}, combined with similar features in the loss function and the forecasts.

Comparing risk profiles for decisions suggested by our Sequential Emulation as being optimal with risk profiles for decisions that a policy maker may already be considering seriously will be a very important decision support tool. We show the risk profile for $\theta_{t_0} = 1$ (corresponding to not abating BAU emissions today at all) under λ_{t_1} in figure 4.34. A curious feature of the two risk profiles we have provided here is that the classically optimal decision (that which minimizes expected loss) has a longer tail than the option of not acting at all. That is, if we do act in the way suggested by expected loss, we risk much larger losses than if we do not act at all. Whilst this is balanced by the thickness of the respective tails so that the expected

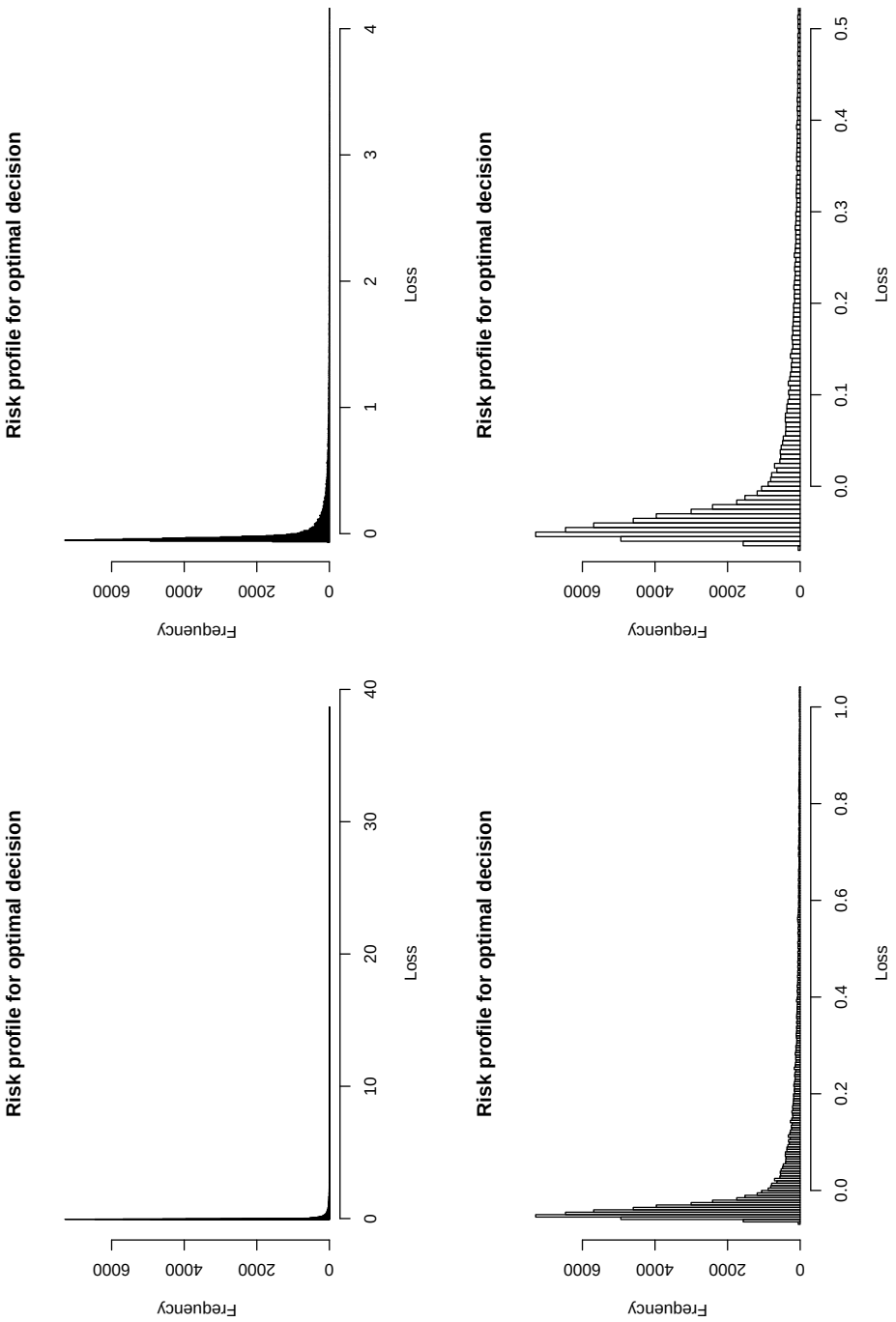

Figure 4.33: Four histograms showing the risk profile for $\theta_{t_0} = arg\min_{\theta_{t_0}} \{\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]\}$ under our original Sequential Emulation (before pruning). Each panel shows an increasingly narrow subset of the possible losses in the risk profile.

loss for not acting is higher, being able to show a policy maker the risk involved in choosing the optimal decision is very useful decision support.

4.7.4 Sensitivity analysis

We perform a limited sensitivity analysis for our example by exploring the effects of our discrepancy beliefs and our particular parametrization of DICE on the results of our Sequential Emulation. Figure 4.35 shows the emulator for the upper bound on expected loss under three different parametrizations of DICE. The central plot is the parametrization we have been working with. The upper-most plot represents a parametrization in which the social welfare function is such that we care more for the wealth of future generations than previously; this is achieved by setting $r = 0.005$ where r is defined as on page 97. The plot at the bottom shows a parametrization where abatement is more expensive initially; achieved by setting $g^b(0) = 0.35$, where the effect of this parameter on abatement cost is presented on page 117. We see that, for our example, the results of Sequential Emulation and therefore the nature of the decision support offered is completely different depending on the parametrization of the loss model DICE. Without any trouble we have found reasonable parametrizations that lead to results recommending totally opposite optimal policies today. This illustrates how important it is to have a very good loss model and for policy makers to think very carefully about their social welfare functions.

Using Sequential Emulation to illustrate the various types of loss surface we obtain under different parametrizations of the loss model will be valuable decision support for policy makers. If the surfaces change dramatically, as they have in our example, they may have to re-visit the models or at least spend further time and resources on encouraging agreement between scientists and their opinions regarding plausible parametrizations. On the other hand, if the surface hardly changes under different parametrizations, we may accept the decision support as robust or, at least, direct resources to other areas, such as research on data collection for the complex system and the computer model for it.

Inclusion of the discrepancy, η, between the computer model and the real-world system must be seen as a key differentiating feature of our methodology. We explore

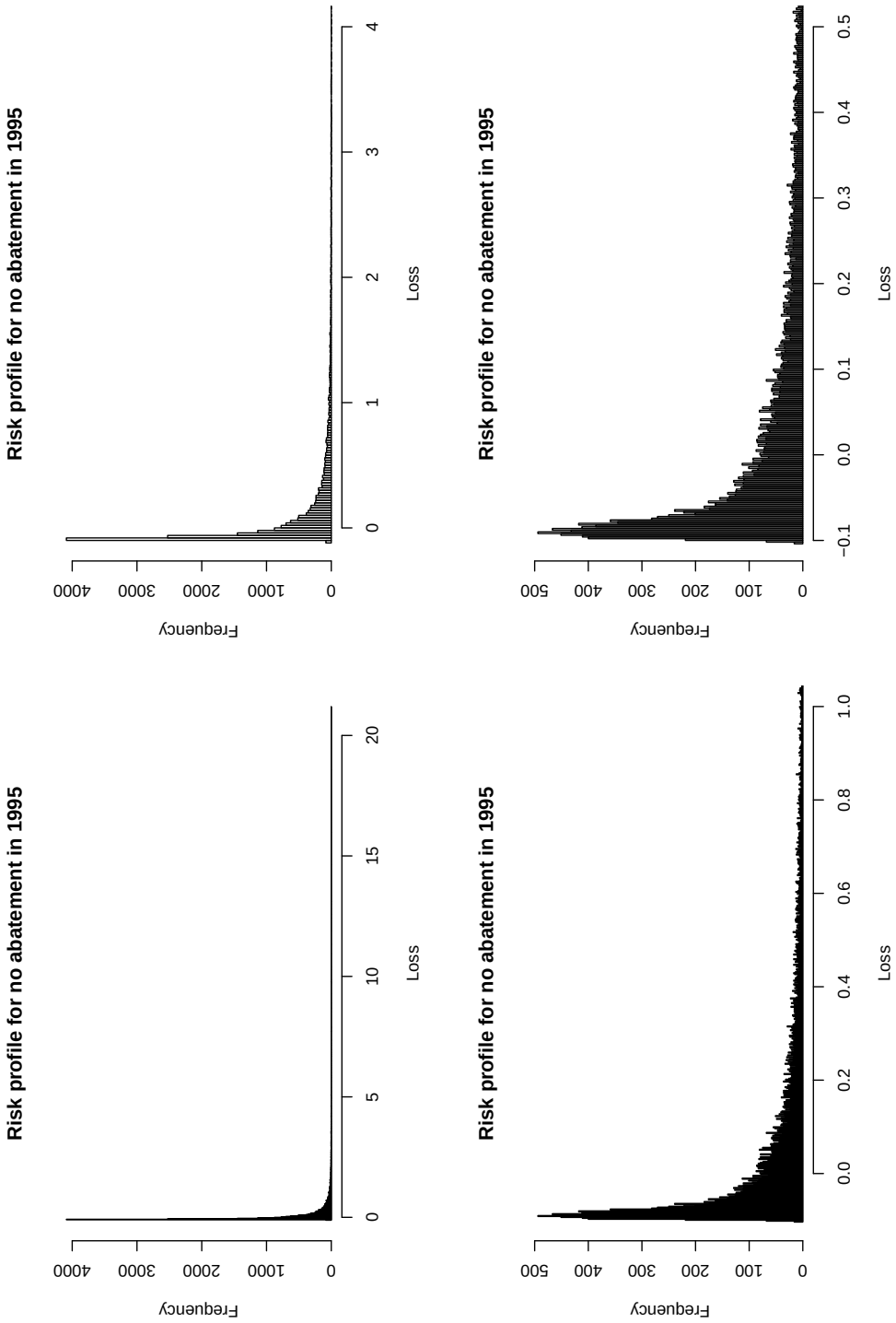

Figure 4.34: Four histograms showing the risk profile for $\theta_{t_0} = 1$ under our original Sequential Emulation (before pruning). Each panel shows an increasingly narrow subset of the possible losses in the risk profile.

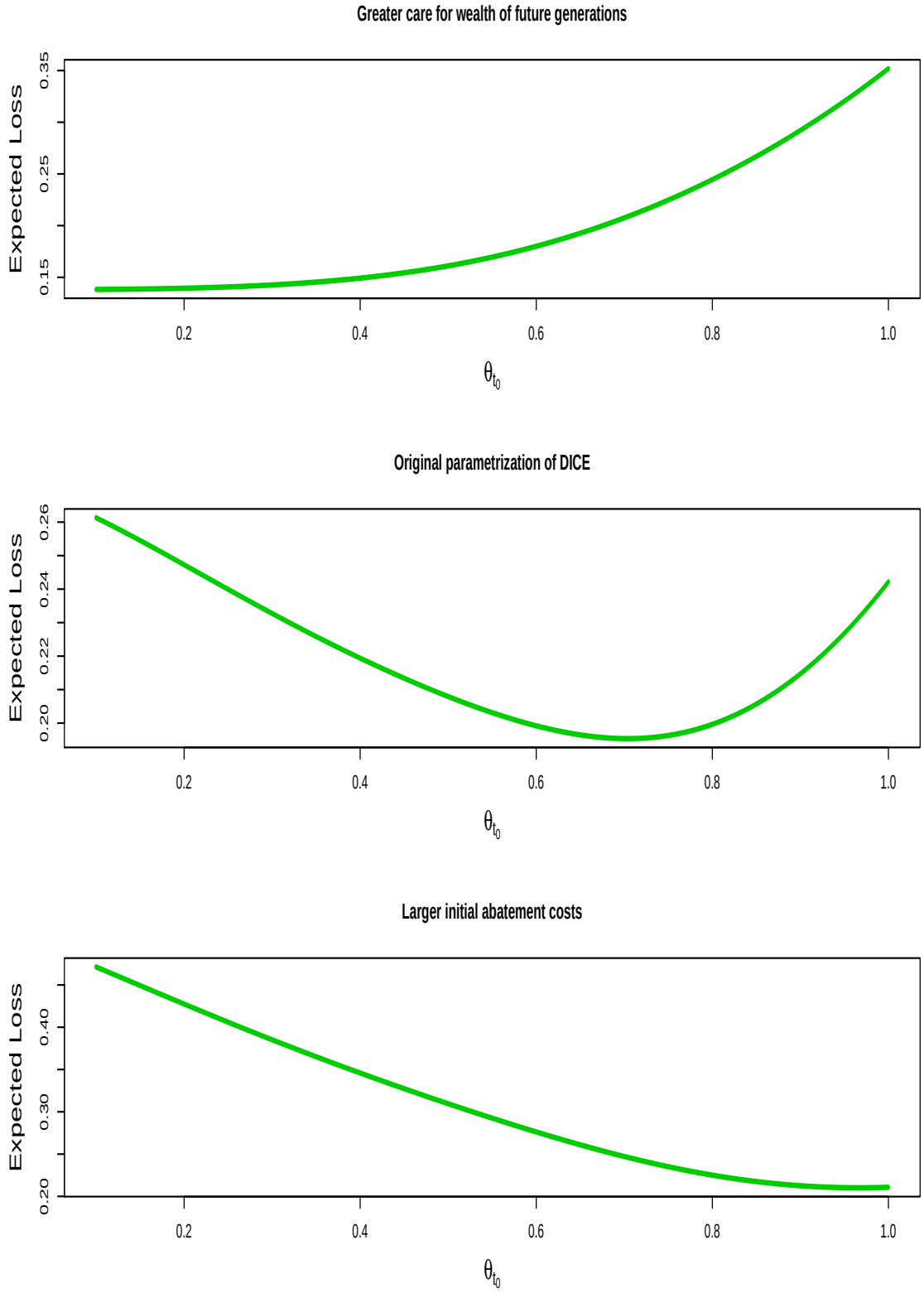

Figure 4.35: Results of Sequential Emulation under three different parametrizations of DICE. The central plot represents our original Sequential Emulations. The first plot has $r = 0.005$ and the final plot has $g^b(0) = 0.35$.

the effect of alternative judgements regarding the size of its variance here. For each of the explored parametrizations of DICE, we performed an alternative Sequential Emulation with very small discrepancy, where standard deviations were divided by 10, and one with larger discrepancy with standard deviations doubled. Inflating them much further would result in impossible future temperatures seen as 'perfectly reasonable' by our statements of uncertainty about y.

We present the results of these Sequential Emulations alongside the results for the unaltered discrepancy in figures 4.36, 4.37 and 4.38. What is striking from these plots is that the curves hardly change, irrespective of how useful we believe C-GOLDSTEIN is for describing the complex system. For the large discrepancy, we would expect a greater difference as the emulator for C-GOLDSTEIN should have less bearing on the image of the loss surface. There are two reasons why this does not happen. Firstly, we do not observe enough data, either now or before adapting our strategy, to greatly change our expectations on future discrepancy. Secondly, our historical climate data is very close to $E\left[f_{t_0}(x^*)\right]$, which means that our expectation for temperature $y_{t_1}(\theta_{t_0})$ will be very close to $E\left[f_{t_1}(x^*, \theta_{t_0})\right]$. These two facts combined mean that even surprising observations, z_{t_1}, will not adjust $E\left[\eta_{t_2}(\theta_{t_0}, \theta_{t_1})\right]$ by a large amount, meaning our expectation for the system reverts to our expectation for the simulator at the best input.

In a case study, when addressing this problem for real, this mean-reversion is unlikely to occur. As we have mentioned before, to make a climate intervention based on only one observation from one year would be foolish; we would expect to have many observations over many years. These combined would appropriately adjust our expectations on the discrepancy so that expectation on future temperatures would divert from those predicted by the model if we had observed an unexpected trend in temperature changes. If, however, it turned out that our loss surfaces remained relatively unchanged for larger or smaller discrepancy variances, and it turned out that this was not an artifact of the mean-reversion we have discussed, but of the loss function itself, then we might judge that the current model for the system was adequate and that no further improved models were required. On the other hand, vastly different curves under different discrepancy variances could

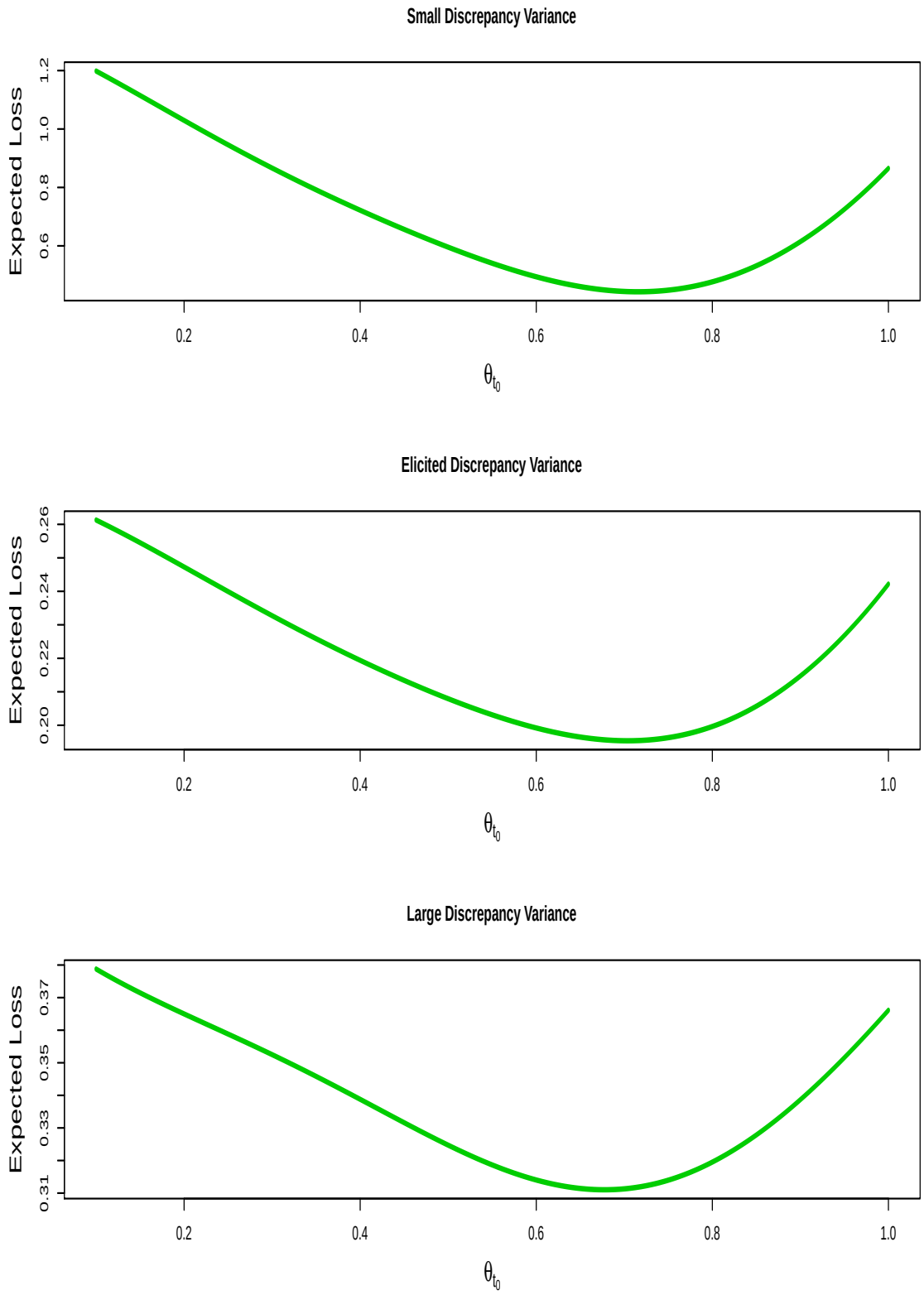

Figure 4.36: Results of Sequential Emulation with three different discrepancy variances for our original parametrization of DICE.

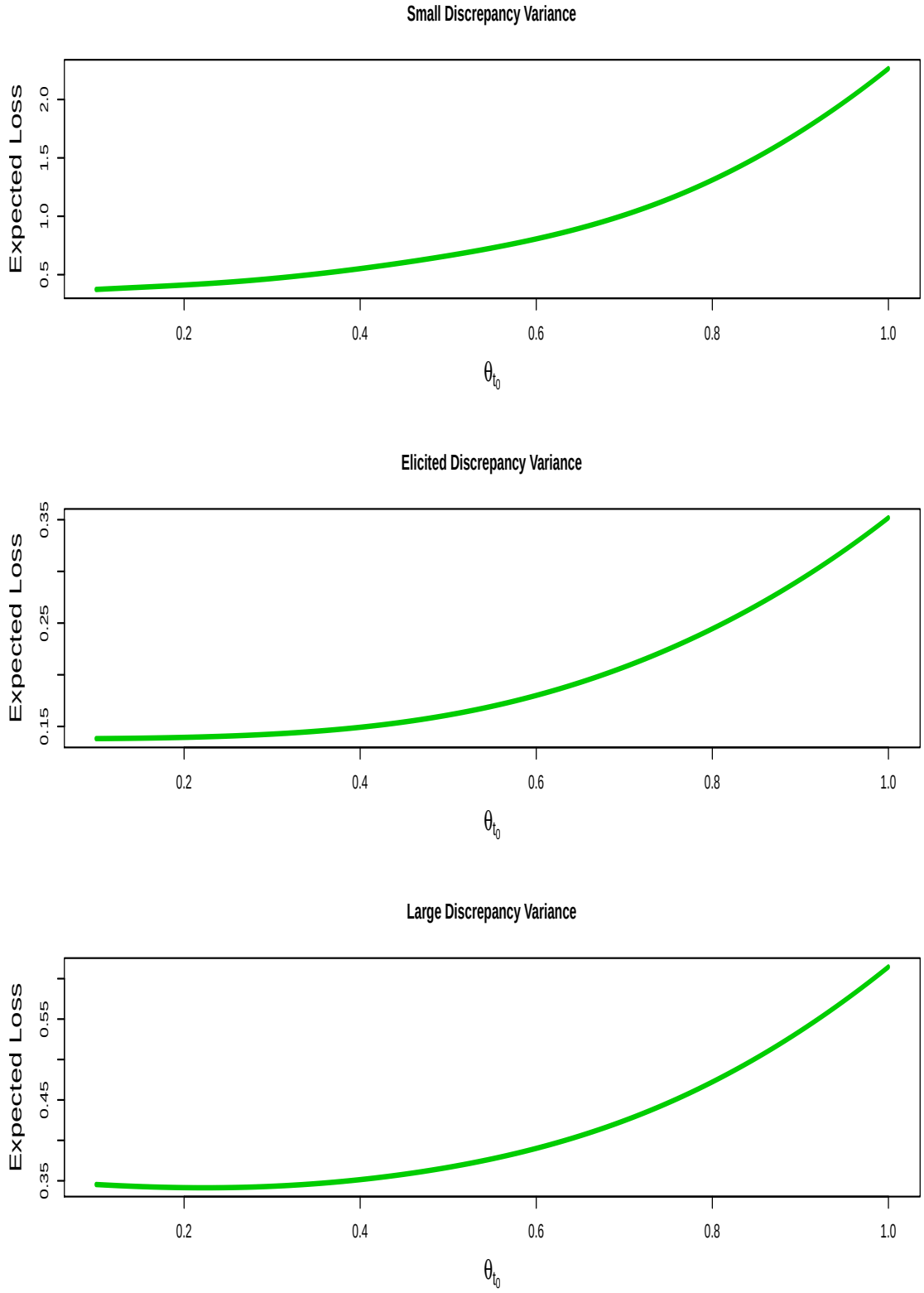

Figure 4.37: Results of Sequential Emulation with three different discrepancy variances for our parametrization of DICE with higher utility for the wealth of future generations.

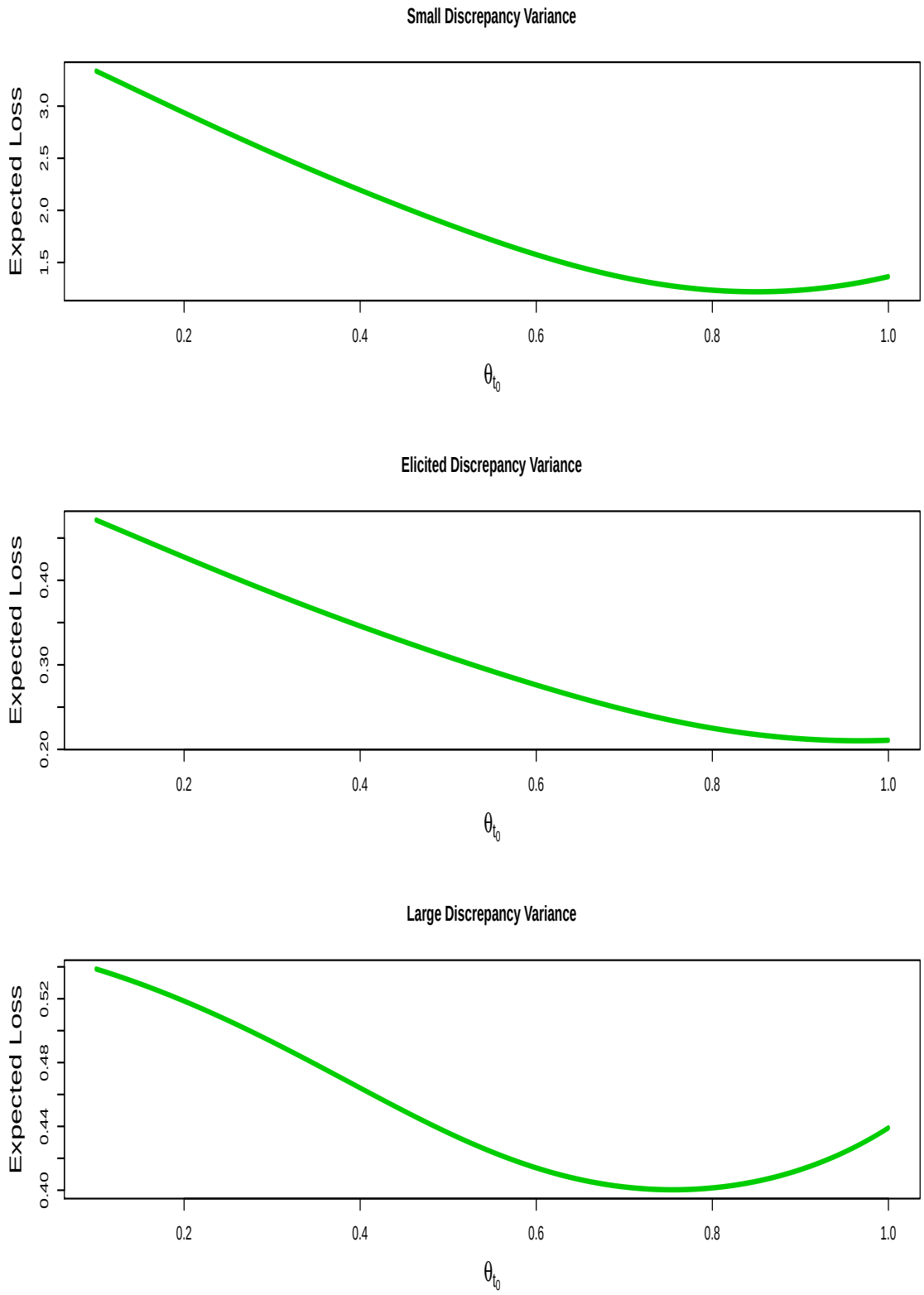

Figure 4.38: Results of Sequential Emulation with three different discrepancy variances for our parametrization of DICE with large immediate abatement costs.

indicate that either more careful research into the discrepancy for our current model was required, or that we needed to resolve as much of the discrepancy variance as possible by building a better computer model.

4.7.5 Relating expected loss to cost

A particularly valuable form of policy support gives the policy maker a rough idea of the relation between his expected losses, which have no scale, and monetary cost, which is well understood. If it turned out, for example, that the difference in monetary cost between the policies that maximised and minimised expected loss was only a few dollars, then the policy maker would be unlikely to be interested in investing in any further exploration of the decision tree and would make his decision. If, on the other hand, our methods predict an expected loss of, say, 0.15 for a particular decision, but that we have a standard deviation of 0.005 for where that expectation is and, moreover, the difference in monetary cost between a loss of 0.15 and 0.16 is trillions of dollars, the policy maker would be very keen to resolve this uncertainty.

The nature of our loss function makes such direct comparison very difficult in this example. A monetary cost is never calculated directly and then converted into a utility. What actually happens is that a per capita consumption is worked out for each decade under a particular abatement scenario with given future temperatures. The logs of these are then combined in a weighted sum designed to reflect society's preference for consumption today as opposed to in the future. We then compute a utility for the wealth of future generations using (4.1), and add this.

To provide an order of magnitude monetary comparison of expected losses, we can look at the difference in global consumption at each decade for two outcomes with different expected losses. The idea is to choose a particular value of θ_{t_0} and perform a forward sampling experiment where, for each sample, we record the time series of global consumptions. Following such an experiment, we can compare the consumption series for different sampled losses. Figure 4.39 shows such a comparison. We conducted a forward sampling experiment at the 'optimal' value of θ_{t_0} (according to our emulator), and recorded the consumption series. We compare two

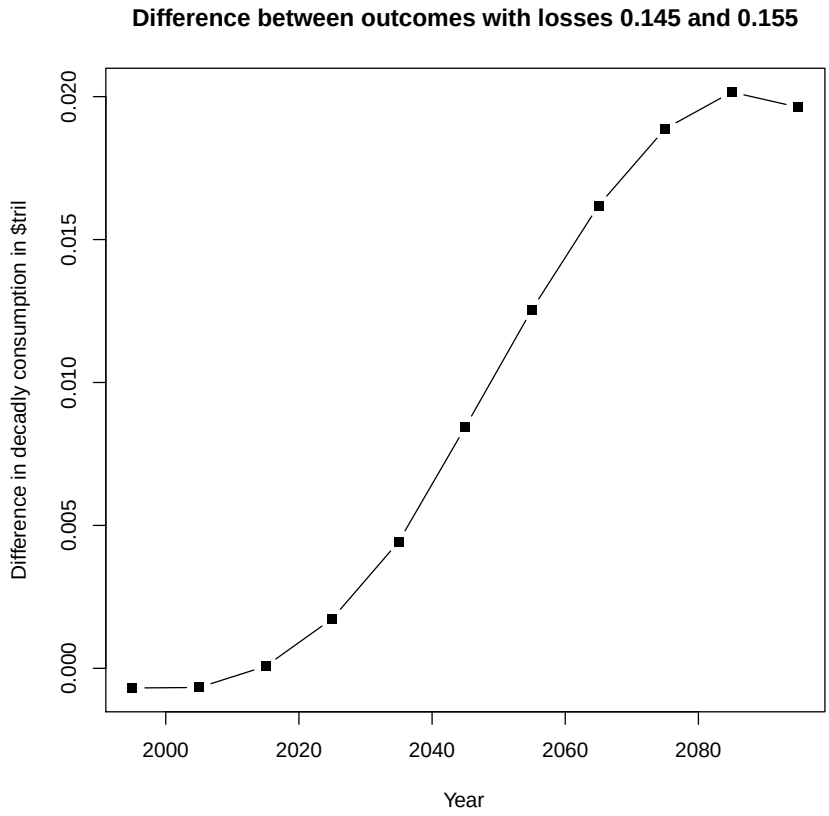

Figure 4.39: A plot of the annual differences in global consumption for two forward samples taken from $\theta_{t_0} \approx 0.7$. The first has expected loss of 0.145 and the second has expected loss 0.155.

consumption series for expected losses that were roughly 0.01 utility units apart. The graph shows that, given this abatement strategy, the decadal difference in consumption for an expected loss 0.01 utility units higher was up to $20 billion.

This may seem like a small figure on the global scale, but may be significant if a small handful of countries (for example, in Africa) had to endure most of the loss. Although this is only an order of magnitude diagnostic, it does show that a difference in expected loss of 0.01 is worth worrying about and may, therefore, lead to further pruning and fine tuning of our emulators.

Chapter 5

Policy support with evolving computer models

In this chapter we consider a more general form of the policy problem than the one considered so far. In this version of the problem, we consider making policy today when we know that our reaction to future observations of the complex system may be influenced by improved versions of our current computer model. Section 5.1 provides an introduction to the concept of evolving models. Section 5.2 describes the current technology for relating a collection of models to each other and to the complex system via reification. We define the more general policy problem in section 5.3 and provide an example for how one might relate a number of similar computer models using structural reification in section 5.4.

In section 5.5 we describe the issues involved in considering runs on future models as part of our decision making process, and introduce an alternative approach to quantifying observable information on future models. In section 5.6 we use this approach to define forecasts that are appropriately influenced by observed information regarding improved models. In section 5.7 we present a version of the Sequential Emulation algorithm that appropriately handles potential future information from improved models. We provide an illustration of these methods in section 5.8 using the same models as seen in chapter 4. Section 5.9 offers a discussion of these ideas.

5.1 Evolving computer models

One of the principle features of our Sequential Emulation methodology is that our policy support is offered with consideration to the way our strategies will adapt when faced with future observations of the system. Underpinning this is the idea that we know how we should react given any particular observation. If the assumption that we made in section 2.1.2, namely that our current simulator, $f(x, \theta)$, is our 'best' simulator and that we will not have access to any other at any time in the future, is true, then the probabilities obtained through our decision-dependent forecasts will always represent our beliefs in the future. This means that we know how we should react to any future observations because our beliefs combined with the loss function determine an expected utility. For a real-world policy problem, this assumption is unlikely to be true because computer models evolve.

There are many reasons that computer models for a complex system evolve. As computer technology improves, the code may be solved at higher resolutions. This would allow smaller scale effects that perhaps were not captured by the current solver to influence the output. It may be that certain aspects of the model were not constructed in as much detail as the scientists would have liked due to budget or technological constraints. The atmosphere and sea ice components of C-GOLDSTEIN are an example of this. The ocean modelling in C-GOLDSTEIN is very detailed. However, simplified physics is used to describe sea ice and the atmosphere so that the model can be run more quickly on the machines for which it was built. Improved technology, including faster computers, would allow knowledge of the physics to be modelled more accurately whilst maintaining reasonable run times. As scientific understanding of the behaviour of the system improves, the models will evolve to reflect that knowledge. It is, therefore, unrealistic to expect our downstream intervention strategies to be made using our current model when we know that we shall be able to use improved models, as well as the system observations, to help adapt policy.

Knowing that the computer model we have now will have been replaced by an improved version when we consider adapting our strategy has important implications for the Sequential Emulation methodology. Not only must our beliefs change in the

future when we observe the system, but they will change when observing runs on improved models. The way in which these beliefs will change depends on how we believe any improved model relates to our current model and how all of these models relate to the complex system $y(\theta)$.

5.2 Reification

5.2.1 The best input re-visited

In section 2.1.2 we introduced the best input assumption for our current simulator $f(x, \theta)$. This stated that there exists x^* such that

$$y(\theta) = f(x^*, \theta) + \eta(\theta) \tag{5.1}$$

with $\eta(\theta)$ independent of x^* and of $f(x, \theta)$ for any x and θ. Suppose now that at time t_1, not only will we observe aspects of $y_{t_1}(\theta_{t_0})$, but we will have access to runs on an improved version of f, denoted $f^{'}$. We must now relate $f^{'}$ to the system in order to know how we might react to our system observations in the future. We suppose, without loss of generality, that the improved model has model inputs x and decision inputs θ. If the improved model has processes with required model input parameters that are not present on our original simulator, we may simply extend x on both simulators.

The most natural way to link $f^{'}$ to y is via another best input assumption, namely that there is an x_0^*, such that

$$y(\theta) = f^{'}(x_0^*, \theta) + \eta^{'}(\theta), \tag{5.2}$$

with $\eta^{'}(\theta)$ independent of x_0^* and of $f^{'}(x, \theta)$ for any x or θ. After all, if we accept that our current model has such a relationship with the system, we would normally expect an improved version of the model to have such a property, albeit with a smaller discrepancy. The two models, f and $f^{'}$, would normally be closely related. Often $f^{'}$ will have evolved directly from f and might contain some identical elements of code. It would be strange to consider that f had an x^* such that $f(x^*, \theta)$ was

sufficient for any information about the system contained in the model, but that an improved, but closely related, model did not.

By stating that $f^{'}$ is an improvement on f and making assumptions (5.1) and (5.2), we are stating that if we knew x_0^* and $f^{'}$, the values of x^* and $f(x^*, \theta)$ would not reveal anything new about $y(\theta)$. This sufficiency statement may be written

$$y(\theta) \perp\!\!\!\perp \{x^*, f\} | \{x_0^*, f^{'}\}. \tag{5.3}$$

From (5.1) and (5.2) we can write

$$\eta(\theta) = f^{'}(x_0^*, \theta) - f(x^*, \theta) + \eta^{'}(\theta) \tag{5.4}$$

so that the difference between the two simulators at their respective best inputs represents modelling of the discrepancy for our current model. From (5.2) we know that $\eta^{'}(\theta) \perp\!\!\!\perp \{x^*, f\}$ and, because $\eta(\theta) \perp\!\!\!\perp \{x^*, f\}$, we deduce from (5.4) that

$$f^{'}(x_0^*, \theta) - f(x^*, \theta) \perp\!\!\!\perp \{x^*, f\}.$$

If we consider improved models built by the same scientists that built our current model or, indeed, any model evolving directly from our current model, this relationship seems counter intuitive. Unless the models are completely different, one would expect that knowing the location of x^* would be informative for the location of x_0^*. We would also expect that knowing the value of $f(\hat{x}, \hat{\theta})$ for some $\hat{x}$, $\hat{\theta}$, would tell us something about $f^{'}(\hat{x}, \hat{\theta})$. Hence we expect that knowing x^* and $f(x^*, \theta)$ would be informative for $f^{'}(x_0^*, \theta)$ and so for $f^{'}(x_0^*, \theta) - f(x^*, \theta)$.

Unless we genuinely believe that x_0^* is not related to x^* and that $f^{'}$ is not related to f then, we cannot use a best input assumption for both models unless we are prepared to adopt a very restrictive form of our beliefs. We must, therefore, either specify a joint distribution over $\{\eta(\theta), f, x^*\}$ or over $\{\eta^{'}(\theta), f^{'}, x_0^*\}$. To ignore this issue would mean providing policy support that was based on beliefs that we knew we would not hold when adapting strategies downstream.

In order to avoid these problems we must have, at most, one model satisfying a best input assumption. We must, therefore, specify a joint distribution over $\{\eta(\theta), f, x^*\}$ or $\{\eta^{'}(\theta), f^{'}, x_0^*\}$, or else reject the notion of a best input entirely. We

are reluctant to abandon the best input approach completely, because without the assumption it would be very difficult to relate any runs made on our models to the system. Specifying a joint distribution over either $\{\eta(\theta), f, x^*\}$ or $\{\eta^{'}(\theta), f^{'}, x_0^*\}$ is not only very difficult, but is laden with implications for the wider methodology. To perform Sequential Emulation we must be able to sample from distributions such as $p(y|z^m, \theta^m)$ and $p(z_{t_j}|z^{j-1}, \theta^{j-1})$ for $1 < j \leq m$. These must rely on the relationship between the system and the improved model and any potential runs made on it. Furthermore, we may have access to even better models at later time points. It is conceivable that we have a new improved model at every one of our m intervention points!

In order to present a general methodology for providing policy support for these types of problems, then, we require a general framework for handling improved models and relating them to the system. It is worth commenting that even in other applications, where we may not be considering building future models, consideration of the potential changes arising from improved versions of models is one of the most useful ways to elicit beliefs about model discrepancy. Therefore, we require a coherent and consistent structure for handling the system and it's relation to our current model and potential improvements to it. We turn to the concept of reification, introduced by Goldstein and Rougier in [40].

5.2.2 The Reified Simulator

The idea introduced by Goldstein and Rougier [40] is to think about an idealized simulator, the *reified* simulator, that satisfies a best input assumption and is such that we cannot at present conceive of any way in which the reified simulator may be improved. In other words, we imagine a simulator that includes the careful modelling of any feasible improvements to our current simulator that we can think of today. For example, our current simulator may ignore or approximate certain processes that we imagine could be included on a future simulator. We may also know that our mathematical description of the system models processes that our solver cannot capture, but that a better solver would. The reified simulator must be such that if we were told of an even better simulator, we could not think of any way in which it

would differ from the reified simulator. Put another way, if we were to consider the difference between the reified simulator and a supposedly more accurate simulator at their best inputs, we would consider this difference to be a mean zero random process.

Suppose that the reified simulator is $f^*(x, \theta)$, where x is a vector containing the model inputs for our current simulator, all model inputs from any of the possible improved simulators we are currently considering and, perhaps, other inputs only found on the reified simulator. We then judge that there exists an x^* such that

$$y(\theta) = f^*(x^*, \theta) + \eta^*(\theta), \tag{5.5}$$

with $\eta^*(\theta)$ independent of f, f^*, x^* and runs on any improved version of f, for all θ. Note that we now do not have a best input assumption for any of the other models under consideration, including our current simulator. This means that x^* does not have the same meaning it had in earlier chapters, namely as a best input for our current model; it is simply the best input for the reified simulator.

The reified simulator is judged to separate the system from all current and future versions of the model that we can imagine. This means that runs on any model for the system are informative for the system precisely because they are informative for the reified simulator and for no other reason. A pragmatic way of considering the reified form, is to think of it as an 'upper bound' for all models achievable in the time frame of our analysis. A detailed discussion regarding the the practicalities and foundational implications of introducing the reified simulator can be found on page 9 of [40] and in the discussion and rejoinder of that same paper. We do not repeat any of that discussion here. However it is worth adding to that discussion in the context of the policy problem.

In many applications of the analysis of computer experiments, the possibility that there will be better simulators in the future may not deter the analyst from using a best input assumption on his current simulator. Even if he knows that such an assumption restricts his belief specification in the future, or if he intends to ignore temporal coherence, he may still view the best input assumption as a pragmatic 'way in' to a very hard problem. There may be other, more pressing, concerns in certain problems than considering what one might do in the future with

a better simulator. In the policy problem we have no such luxury. To offer decision support for policy that must be made today, we have to use all of our knowledge regarding how we shall behave in the future. This means considering how improved models and observations of the real world that we know we will be able to see affect our decisions today.

Consider, for example, the CO_2 abatement problem. Suppose we ignore the possibility that we shall have improved models in the future. This may lead us to consider an analysis such as that given in chapter 4, leading to abatement of, say, 20% per decade. If we knew that in ten years we would be able to see runs of the most powerful climate model ever, one that resolves almost all of the uncertainty in the current discrepancy, would we not be more inclined to abate less now and wait for the results of the analysis of that model? Furthermore, if an analysis showed that our ideal policy was completely insensitive to any observations on a new and better model, that would tell us that we need not build the new one at all!

As was discussed in section 3.2, policy makers are wary of using computer models to aid decision making and one of the reasons for this is that the models are always changing (see, for instance, Brugnach et al [13]). Each improved version of a computer model may help a policy maker, but he does not know whether to trust his current one and make a very proactive policy or to wait for a better simulator and make a safer (at least politically or in terms of initial cost) policy today. By introducing a coherent belief framework across all improved models that we can think of today, such as that provided through the reified structure described here, we formalize this aspect of the policy problem. Therefore, the decision support we can offer will be based on the knowledge of having access to potential improved models in the future and may even help aid research investment decisions. We will show, for example, that, if required, the decision of whether or not to invest in the building of a particular improved model and how much to invest in running such a model if built, can be included as part of a policy decision for today. We see this application as potentially being particularly valuable in areas like climate change. The building of new climate models, for example, is enormously expensive and any insight into the value of potential gains that can be made by building the model,

before it is actually constructed, could be very valuable.

Having introduced the concept of the reified model, we now restate the policy problem so that the possibility of evolving simulators for the system is handled. We go on to describe how we can use the reified form to provide policy support within this framework.

5.3 The policy problem with evolving simulators

A policy maker must make a decision today, parametrized by θ_{t_0}, in order to influence future states of a complex system. To aid him, he may make runs on a computer simulator $f^0(x, \theta)$ and has observations of the system z_{t_0}. At m different points in the future, $t_1, \ldots, t_m$, the decision maker may observe the system under his current policy and adapt his strategy by making new policy θ_{t_i}, for $i = 1, \ldots, m$. Before making each policy revision the decision maker will have access to a version of the computer model, denoted $f^i(x, \theta)$, that is seen as an improvement to $f^{i-1}(x, \theta)$ for $i = 1, \ldots, m$. Here, x is the vector of all of the inputs that we can possibly conceive of introducing on any version of the simulator, including an idealized version that is better, even, than $f^m(x, \theta)$.

We have a loss model, $L(x_L, y(\theta), \theta)$, as defined on page 47, and define z^k and θ^k for $k = 1, \ldots, m$, as on page 53. Our loss model should incorporate the cost of constructing and running new models. We define

$$f^{[k]}(x, \theta) = f^0(x, \theta), f^1(x, \theta), \ldots, f^k(x, \theta)$$

for $k = 1, \ldots, m$. We assume that each simulator has a vector valued output that corresponds to the system, $y(\theta)$, at all time points under consideration. Therefore, the subvector $f^k_{t_f}(x, \theta)$ of the simulator we will have at time t_k has the same units and components of $y_{t_f}(\theta)$.

In order to provide policy support for this problem, we must specify how each of the simulators are related to each other and how they are related to the system. We use reification to provide the link to the system and, therefore, introduce a reified simulator, $f^*(x, \theta)$, such that there exists x^* with

$$y(\theta) = f^*(x^*, \theta) + \eta^*(\theta), \tag{5.6}$$

and $\eta^*(\theta)$ independent of x^*, $f^*(x,\theta)$ and $f^{[m]}(x,\theta)$ for all x and θ. Each of the models is informative for $y(\theta)$ because it is informative for the reified simulator. We illustrate the different relationships for the problem we have described using the example influence diagram in figure 5.1.

We now link each of the simulators to each other and to the reified simulator.

5.4 Relating simulators

There may be many ways of specifying our beliefs across the collection of simulators that we have introduced. The one we consider in this thesis, and arguably the most natural method of belief specification for this problem, involves constructing emulators for each of the models and linking the processes and coefficients in the emulators. This method is what Goldstein and Rougier [40] call *structural reification.*

Suppose that we have already used the methods of chapter 2 to build an emulator for $f^0(x,\theta)$ so that we have

$$f_i^0(x,\theta) = \beta_{ij}^0 g_j(x,\theta) + u_i^0(x,\theta), \tag{5.7}$$

as usual. We now consider the improved simulator, $f^1(x,\theta)$ to be

$$f_i^1(x,\theta) = \beta_{ij}^1 g_j(x,\theta) + \alpha_{ij}^1 h_j^1(x,\theta) + u_i^1(x,\theta)$$

where β_{ij}^1 are related to β_{ij}^0 in some way, and the $h_j^1(x,\theta)$ represent new regression terms only present on the emulator for the new model. The residual process $u_i^1(x,\theta)$ has some covariance with $u_i^0(x,\theta)$. We might often judge $u_i^1(x,\theta)$, α_{ij}^1 and β_{ij}^1 to be independent of each other.

This is a natural and flexible specification, allowing us to write down our beliefs about the behaviour of any new processes captured only on the new model through $h_j^1(x,\theta)$ and $u_i^1(x,\theta)$, whilst also allowing us to describe similarities in the models through $Cov\,[\beta^0,\beta^1]$ and $Cov\,[u_i^0(x,\theta), u_i^1(x,\theta)]$. If we were to consider an emulator for each of our $m+2$ models in this way; that is, as an emulator with some extra regression terms and covariances across the coefficients of shared terms and processes, the notation that we have introduced above may become quite cumbersome. Instead, we allow $g(x,\theta)$ to be a vector containing all of the regression terms that

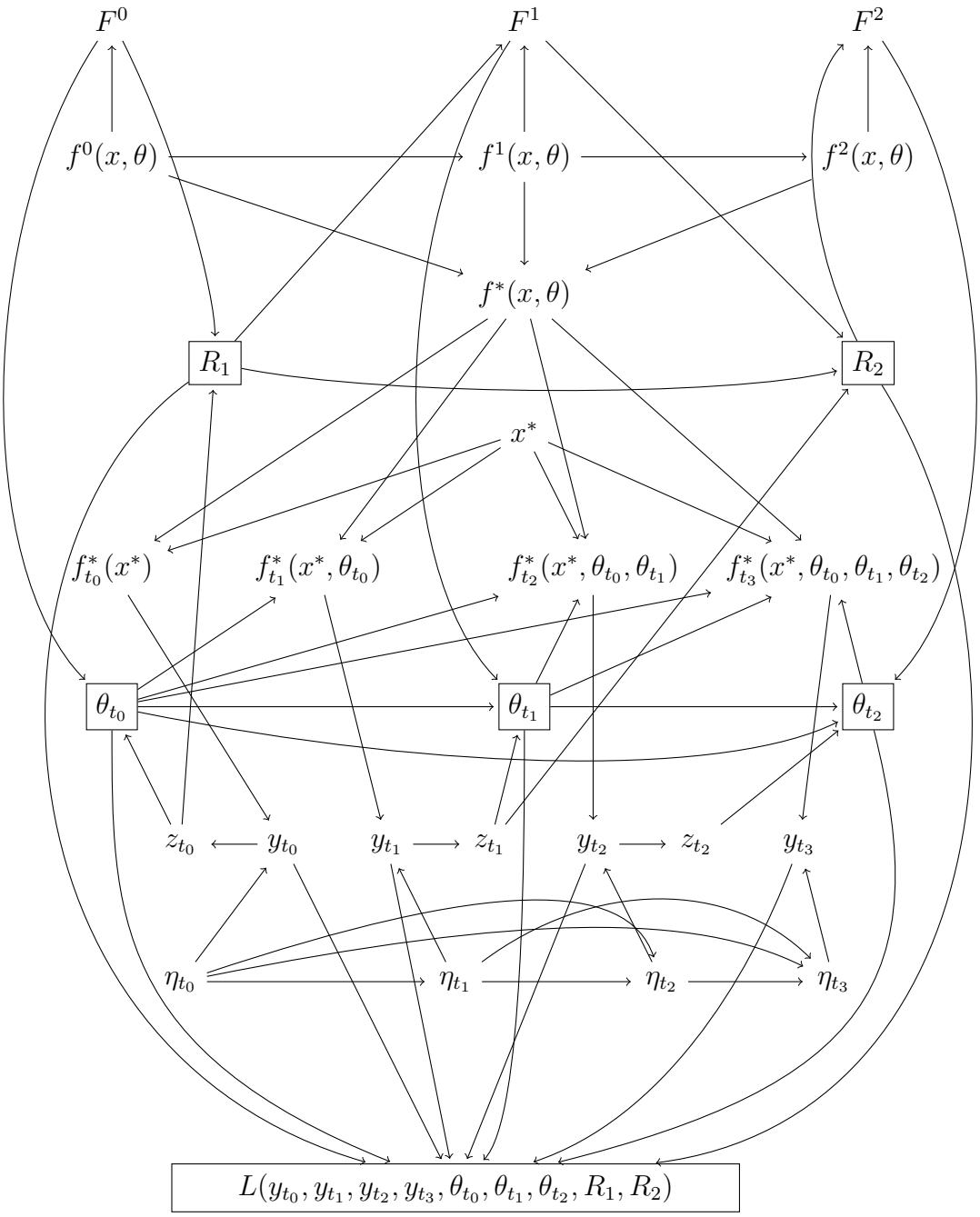

Figure 5.1: Our statement of the decision problem and the modelling statements we have made define an influence diagram. This figure presents an example of this influence diagram for the case where we have 2 downstream intervention points at times t_1 and t_2, and the option to build and run an improved version of our current model at each of these times. To make the research investment decisions clear on the diagram, we give them separate nodes and label them R_1 and R_2. These represent the decisions of whether or not to build and how much to run each new model.

we are considering for inclusion in any of the emulators, for any of our models. We then specify that

$$f_i^k(x,\theta) = \beta_{ij}^k g_j(x,\theta) + u_i^k(x,\theta), \tag{5.8}$$

with each coefficient matrix β^k extended so that it is compatible with the enlarged $g(x,\theta)$. Any coefficients for regression terms that are not present on a particular emulator are given expectation and variance equal to zero on that emulator. We also specify that

$$f_i^*(x,\theta) = \beta_{ij}^* g_j(x,\theta) + u_i^*(x,\theta). \tag{5.9}$$

To link each of our models to the system we must specify beliefs over the collections $\{\beta^0, \beta^1, \ldots, \beta^m, \beta^*\}$ and over $\{u^0(x,\theta), \ldots, u^m(x,\theta), u^*(x,\theta)\}$ for any x and θ.

Assuming that we have specified beliefs over the collections $\{\beta^0,\beta^1,\ldots,\beta^m,\beta^*\}$ and $\{u^0(x,\theta), \ldots, u^m(x,\theta), u^*(x,\theta)\}$, we may obtain decision-dependent forecasts in precisely the same way as we described in section 2.3.2, where β and $u(x,\theta)$ are replaced with β^* and $u^*(x,\theta)$. Runs observed on any of our models update β^*, the random field $u^*(x,\theta)$, and the covariance between the two, through the structure imposed on $\{\beta^0, \beta^1, \ldots, \beta^m, \beta^*\}$ and $\{u^0(x,\theta), \ldots, u^m(x,\theta), u^*(x,\theta)\}$. The best input, x^*, is then integrated out of our expression for $y(\theta)$, giving an expectation and covariance structure on $y(\theta)$. This is then adjusted by any system observations as normal.

Provided that we can link the coefficients and residual processes across model emulators, then we have a way of assimilating collections of runs on any of our models and calculating decision-dependent forecasts for subsets of $y(\theta)$. Before addressing the decision problem directly, we introduce a natural and flexible way of linking coefficients and emulator residuals across models. The method described is not the only possible way of linking these quantities probabilistically, but it will be useful to have a specified framework in mind in order to illustrate some of our methods of decision support.

5.4.1 Relating emulators

We suggest the following framework for producing covariances across the coefficients of each emulator. We let

$$\beta_{ij}^{k} = \beta_{ij}^{k-1} + b_{ij}^{k} \tag{5.10}$$

with b_{ij}^{k} independent of β_{ij}^{k-1} for all i, j and $k = 1, \ldots, m$. Similarly, we let

$$\beta_{ij}^{*} = \beta_{ij}^{m} + b_{ij}^{*} \tag{5.11}$$

with b_{ij}^{*} independent of β_{ij}^{m}. We also assume that b^{*} and b^{k} are independent of all b^{r} (for all $k \neq r$ and $k = 1, \ldots, m$). To compute $Cov\left[\beta^{k}, \beta^{j}\right]$ for any $k, j \in \{0, 1, \ldots, m\}$ we need only means and variances for the matrices b^{i} for $i = 1, \ldots, \max(k, j)$, as well as $Var\left[\beta^{0}\right]$, which we have already from our emulator for the current model. For example,

$$\begin{aligned} Cov\left[\beta_{ij}^{1}, \beta_{kl}^{3}\right] &= Cov\left[\beta_{ij}^{1}, \beta_{kl}^{2} + b_{kl}^{3}\right] \\ &= Cov\left[\beta_{ij}^{1}, \beta_{kl}^{1}\right] \\ &= Cov\left[\beta_{ij}^{0} + b_{ij}^{1}, \beta_{kl}^{0} + b_{kl}^{1}\right] \\ &= Var\left[\beta^{0}\right]_{ijkl} + Var\left[b^{1}\right]_{ijkl}. \end{aligned}$$

In order to link the emulator residuals we may consider a relation such as

$$u^{k}(x, \theta) = c^{k} u^{k-1}(x, \theta) + \delta^{k}(x, \theta) \tag{5.12}$$

for $k = 1, \ldots, m$ and

$$u^{*}(x, \theta) = c^{*} u^{m}(x, \theta) + \delta^{*}(x, \theta), \tag{5.13}$$

where $\delta^{1}(x, \theta), \ldots, \delta^{m}(x, \theta), \delta^{*}(x, \theta)$ are independent stochastic processes that are also independent of the residual processes on previous simulators. So, for example, $\delta^{k}(x, \theta)$ would be assumed to be independent of $u^{j}(x, \theta)$ for $j < k$. The coefficients $c^{1}, \ldots, c^{m}, c^{*}$ may be known quantities that we specify, or we may also express uncertainties on them. If we were to treat $c^{1}, \ldots, c^{m}, c^{*}$ as unknown quantities, we might specify that they were independent of $\delta^{1}(x, \theta), \ldots, \delta^{m}(x, \theta), \delta^{*}(x, \theta)$.

This specification has the general effect of enlarging our uncertainties on coefficients and residuals as the model improves. However, we are not increasing

the overall discrepancy. The discrepancy for our current model is effectively being structured into parts explained by the better models, in addition to an independent remainder $\eta^*(\theta)$. We may resolve much of our uncertainty regarding $y(\theta)$ through building and running the better models, and can resolve uncertainty about $\eta^*(\theta)$ only by observing the system directly.

5.5 Model-related observations

5.5.1 Future model runs

In addition to the system observations, z_{t_k}, that we see at each time point, t_k, we will observe n_k runs on the new model, $f^k(x, \theta)$, available to us at that time. These runs will be performed according to some design, (Ω^k, Θ^k). Let

$$F^k = \left(f^k(\Omega_1^k, \Theta_1^k), \ldots, f^k(\Omega_{n_k}^k, \Theta_{n_k}^k)\right)$$

for $k = 1, \ldots m$. At each time point we observe z_{t_k} and F^k. Our beliefs about the system, adjusted by z_{t_k} and F^k, would be obtained by adjusting $\{\beta^*, u^*(x, \theta)\}$ by F^k, integrating out x^*, and then adjusting the derived moments of $y(\theta)$ by z_{t_k} as usual. This is feasible, in principle, given a belief structure on $\{\beta^*, u^*(x, \theta)\}$ such as that defined by (5.10), (5.11), (5.12) and (5.13). However, to include the knowledge that this is the way we intend to behave in the future as part of our decision making process today is, in practice, not feasible.

The issue lies in the updating of $\{\beta^*, u^*(x, \theta)\}$ by the random field induced by $u^k(\Omega_1^k, \Theta_1^k), \ldots, u^k(\Omega_{n_k}^k, \Theta_{n_k}^k)$. At this point in time, we do not even know n_k, and although we could fix n_k or make it a decision variable, we do not know what design, (Ω^k, Θ^k), we might use in the future. We could, theoretically, add the design problem to the decision tree. However, the vast difficulty of solving the decision problem for just one design makes this approach totally infeasible. We could decide to use the same design as we had used to make runs on our current simulator. However, this is unlikely to be a good design and considerably restricts what we are able to learn about the reified model from our new simulator.

To consider observations of $u^k(x, \theta)$ as part of our decision making process today

seems impractical, if not impossible. During the reification process we constructed the enlarged vector of regressors $g(x, \theta)$ and our beliefs about β^k so that our beliefs about the surface $\beta^k_{ij} g_j(x, \theta)$ represented all of the possible structure that we could think of modelling as part of $f^k(x, \theta)$. By the nature of this description, we have no idea today what $u^k(x, \theta)$ may be like. If $u^k(x, \theta)$ may take any form (within the bounds set by our covariance structure on the process), it would be impractical to consider the implications of its possible observed future forms for our expected loss and then integrate over everything. If we have beliefs about what the surface will look like in the future, as we do with $\beta^k_{ij} g_j(x, \theta)$, it makes sense to see how our decisions today are altered by changes to those beliefs. Knowing that we will no longer be ignorant about the surface $u^k(x, \theta)$ at time t_k does not affect our decision making today without having beliefs about how our ignorance regarding $u^k(x, \theta)$ might change.

One way we might describe how our ignorance about $u^k(x, \theta)$ may change would be to consider that it's variance will be reduced. To simplify our exposition we do not consider this case and leave it as an interesting area of further work. We therefore consider only those model-related observations in the future about which we have structural beliefs today.

5.5.2 An alternative treatment of future model observations

Observing F^k will change our beliefs about the regression surface on the future model by updating our beliefs about β^k. In addition to this, we know that $Var_{F^k}\left[\beta^k\right]$ does not depend on F^k, but on the design used to generate F^k. This means that the amount of uncertainty we leave unresolved on β^k having observed runs on the new model is a decision that can be made today (or at time t_{k-1}).

Let

$$F^{[k]} = F^0, \ldots, F^k.$$

Then what is observed regarding the regression surface on the new model at time t_k is actually the random quantity $E_{F^{[k]}}\left[\beta^k\right]$. Therefore, future, model-related, observations can be broken down into the observation of $E_{F^{[k]}}\left[\beta^k\right]$ and some information on the residual surface. Note that if we actually observed data F^k at time t_k, we

would want to adjust our prior beliefs about β^k by all runs that we have seen on all previous models. By ignoring the potential for future observation of the residual surface, we can replace the observation of F^k at time t_k with the observation $E_{F^{[k]}}\left[\beta^k\right]$ on our decision tree, as we will now explain. This leaves us with a decision tree that we may give insight into via a modified Sequential Emulation approach.

Define

$$H^k = E_{F^{[k]}}\left[\beta^k\right],$$

then at time t_k, by ignoring the residual surface for $f^k(x,\theta)$, we consider the observation of $\{H^k, z_{t_k}\}$ as a useful surrogate to $\{F^k, z_{t_k}\}$. At each time point, t_k, we have the decision θ_{t_k} to make as usual. In addition to this, we must also choose how much uncertainty will be left in the coefficients of the simulator we will have at time t_{k+1}. In effect, this means choosing how much to run the model. Our loss function must be modified to include the cost of building and running new simulators at each of the intervention points. Without loss of generality, we let θ_{t_k} be the parametrization of both our system intervention policy at time t_k and the amount of variance to be resolved on β^{k+1} at time t_{k+1}, for $k = 0, \ldots, m-1$. The decision θ_{t_m} is defined as before. To simplify our description we do not consider resolving any of the variance of the residual surface by making model runs. An extension of our methods may consider how we might handle and resolve this variance more carefully.

The decision tree for this problem is given in figure 5.2. Note that each observation for time $t_1, \ldots, t_m$ is now the pair $\{H^k, z_{t_k}\}$. In order to perform a Sequential Emulation of this decision tree, we must be able to sample from the distributions $p(y|z^m, \theta^m, H^{[m]})$ and $p(z_{t_k}, H^k|z^{k-1}, \theta^{k-1}, H^{[k-1]})$, where

$$H^{[k]} = H^1, \ldots, H^k$$

for $k = 1, \ldots, m$. To enable sampling from these distributions, we introduce some further statistical machinery.

Let

$$\beta^k = H^k + \omega^k \tag{5.14}$$

for $k = 1, \ldots, m$, where ω^k and H^k are uncorrelated. Note that $E\left[\omega^k\right] \equiv 0$. From

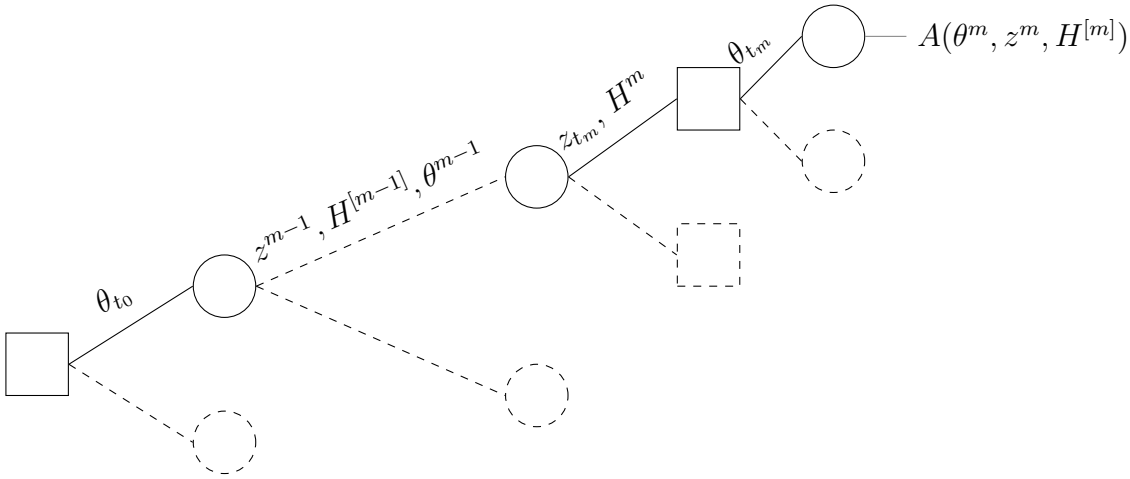

Figure 5.2: This image shows the decision tree for the case where we may build and run new models at each of the m intervention points. Observations of new models are characterized by observed adjusted coefficients, H^k, for the regression surfaces of our m model emulators, for $k = 1, \ldots, m$. The function $A(\theta^m, z^m, H^{[m]})$, defined formally below by equation (5.32), is our expected loss for having taken decisions θ^m and having observed $\{z^m, H^{[m]}\}$.

(5.14) we have $Var\left[\omega^k\right] \in \theta_{t_{k-1}}$, because

$$Var\left[\beta^k\right] = Var\left[H^k\right] + Var\left[\omega^k\right]$$

and

$$Var\left[H^k\right] = RVar_{F^{[k]}}\left[\beta^k\right]$$

from definition (2.11). Hence, definition (2.10) gives

$$Var\left[\omega^k\right] = Var_{F^{[k]}}\left[\beta^k\right],$$

which is part of $\theta_{t_{k-1}}$ under our new notation for this chapter. Note that choosing $Var\left[\omega^k\right] = Var_{F^{[k-1]}}\left[\beta^k\right]$ corresponds to the decision not to build $f^k(x, \theta)$ at all. Strictly speaking, this could mean building the model and not running it. However, we do not consider such highly improbable behaviour and assume that if the model is built, at least some uncertainty on the coefficients must be resolved through running the model.

5.6 Reified decision-dependent forecasts

Our approach to sampling from the required distributions is to characterize them via Bayes Linear decision-dependent forecasts, as seen in chapter 3. We begin by showing how we use Bayes Linear methods to obtain an appropriate mean and variance for the distribution of y given $\{z^m, \theta^m, H^{[m]}\}$. As with all of the Bayes Linear methods of forecasting used so far, we treat information about the model separately from any information about the system. Similarly to the way in which we treated runs F in chapter 2, we use $H^{[m]}$ to update our beliefs about β^* and then integrate x^* out of our expression for f^*. We then adjust the derived beliefs about y by the data z^m. The first quantities required for this calculation are, therefore, $E_{H^{[m]}}[\beta^*]$ and $Var_{H^{[m]}}[\beta^*]$.

5.6.1 Adjusting the reified coefficients

We know that

$$E_{H^{[m]}}[\beta^*] = E[\beta^*] + Cov\left[\beta^*, H^{[m]}\right] Var\left[H^{[m]}\right]^{-1} (H^{[m]} - E\left[H^{[m]}\right])$$

and

$$Var_{H^{[m]}}[\beta^*] = Var[\beta^*] - Cov\left[\beta^*, H^{[m]}\right] Var\left[H^{[m]}\right]^{-1} Cov\left[H^{[m]}, \beta^*\right].$$

The object $H^{[m]}$ represents the collection of m different matrices, $H^1, \ldots, H^m$, each having the same dimensions. For the purposes of these calculations, then, we may treat $H^{[m]}$ as an array with three dimensions so that

$$H^{[m]}_{ijk} = H^k_{ij}.$$

Therefore, we have

$$E_{H^{[m]}}[\beta^*]_{ij} = E[\beta^*]_{ij} + Cov\left[\beta^*, H^{[m]}\right]_{ijklp} Var\left[H^{[m]}\right]^{-1}_{klpqrs} (H^{[m]} - E\left[H^{[m]}\right])_{qrs}$$

and

$$Var_{H^{[m]}}[\beta^*]_{ijkl} = Var[\beta^*]_{ijkl} - Cov\left[\beta^*, H^{[m]}\right]_{ijpqr} Var\left[H^{[m]}\right]^{-1}_{pqrstv} Cov\left[H^{[m]}, \beta^*\right]_{stvkl}.$$

Having specified $E[\beta^*]$ and $Var[\beta^*]$, the objects we require in order to complete this adjustment are $E\left[H^{[m]}\right]$, $Var\left[H^{[m]}\right]$ and $Cov\left[\beta^*, H^{[m]}\right]$. We compute these quantities here assuming relations (5.10) and (5.11). Our aim is to show how one may obtain the desired adjusted moments above, given a sensible form of structural reification used to link the coefficients of evolving models described by (5.10) and (5.11).

The first quantity, $E\left[H^{[m]}\right]$, is trivially derived from the expectations we express for b_{ij}^k, $\forall i, j$ and for $k \in \{1, \ldots, m, *\}$. In order to establish the remaining quantities, $Var\left[H^{[m]}\right]$ and $Cov\left[\beta^*, H^{[m]}\right]$, we shall require further theory from the field of Bayes Linear statistics.

Suppose we have beliefs B that have already been adjusted by data D_1. The *partial adjustment of B by data D_2 given D_1* is written $E_{[D_2|D_1]}[B]$ and is defined via

$$E_{[D_2|D_1]}[B] = E_{D_2 \cup D_1}[B] - E_{D_1}[B]. \tag{5.15}$$

Let

$$\mathbb{A}_{D_1}(D_2) = D_2 - E_{D_1}[D_2],$$

then it can be shown that

$$E_{[D_2|D_1]}[B] = E_{\mathbb{A}_{D_1}(D_2)}[B - E[B]], \tag{5.16}$$

and that

$$Var_{D_2 \cup D_1}[B] = Var_{D_1}[B] - Var\left[E_{[D_2|D_1]}[B]\right]. \tag{5.17}$$

For any vectors D and B, we have the relation

$$Cov[D, \mathbb{A}_D(B)] = 0. \tag{5.18}$$

For further detail regarding partial Bayes Linear analysis see chapter 5 of Goldstein and Wooff [42].

For any integer r such that $k + r \leq m$, we write

$$\beta^{k+r} = \beta^k + \alpha^r,$$

where

$$\alpha^r = \sum_{s=k+1}^{k+r} b^s.$$

We know that β^k and α^r are uncorrelated, by definition. The quantity α^r represents information about the coefficients β^{k+r} that can only be learned by observing runs on any of the models that we might build after time t_k. To ensure this property is true we impose the condition that α^r is also uncorrelated with the residual surface of any emulator for models $f^0(x,\theta),\ldots,f^k(x,\theta)$. Hence we assume that $F^{[k]}$ is uncorrelated with α^r.

Proposition 5.6.1

$$Var\left[H^{[m]}\right]_{ijklpq} = \begin{cases} Var\left[H^k\right]_{ijlp} & q \geq k \\ Var\left[H^q\right]_{ijlp} & q < k \end{cases} \tag{5.19}$$

Proof: In order to establish this relation, we begin by writing

$$Var\left[H^{[m]}\right]_{ijklpq} = Cov\left[H^k_{ij}, H^q_{lp}\right].$$

Suppose, without loss of generality, that $q > k$ and that $r = q - k$. Then

$$Var\left[H^{[m]}\right]_{ijklpq} = Cov\left[H^k_{ij}, H^{k+r}_{lp}\right].$$

Now, by definition we have

$$Cov\left[H^k_{ij}, H^{k+r}_{lp}\right] = Cov\left[E_{F^{[k]}}\left[\beta^k_{ij}\right], E_{F^{[k+r]}}\left[\beta^{k+r}_{lp}\right]\right],$$

and because

$$E_{F^{[k+r]}}\left[\beta^{k+r}\right] = E_{F^{[k+r]}}\left[\beta^k\right] + E_{F^{[k+r]}}\left[\alpha^r\right],$$

we have that

$$Cov\left[H^k_{ij}, H^{k+r}_{lp}\right] = Cov\left[E_{F^{[k]}}\left[\beta^k_{ij}\right], E_{F^{[k+r]}}\left[\beta^k_{lp}\right]\right] + Cov\left[E_{F^{[k]}}\left[\beta^k_{ij}\right], E_{F^{[k+r]}}\left[\alpha^r_{lp}\right]\right].$$

Define

$$D^r = \bigcup_{s=k+1}^{k+r} F^s.$$

Then, by (5.15)

$$\begin{aligned} E_{F^{[k+r]}}\left[\beta^k\right] &= E_{F^{[k]}}\left[\beta^k\right] + E_{[D^r|F^{[k]}]}\left[\beta^k\right] \\ &= E_{F^{[k]}}\left[\beta^k\right] + E_{\mathbb{A}_{F^{[k]}}(D^r)}\left[\beta^k - E\left[\beta^k\right]\right], \end{aligned}$$

so that

$$Cov\left[E_{F^{[k]}}\left[\beta_{ij}^{k}\right], E_{F^{[k+r]}}\left[\beta_{lp}^{k}\right]\right] = Var\left[H^{k}\right]_{ijlp} + Cov\left[E_{F^{[k]}}\left[\beta_{ij}^{k}\right], E_{\mathbb{A}_{F^{[k]}}(D^{r})}\left[\beta_{lp}^{k} - E\left[\beta_{lp}^{k}\right]\right]\right].$$

Now $E_{F^{[k]}}\left[\beta^{k}\right]$ is a linear combination of $F^{[k]}$ and $E_{\mathbb{A}_{F^{[k]}}(D^{r})}\left[\beta^{k} - E\left[\beta^{k}\right]\right]$ is a linear combination of $\mathbb{A}_{F^{[k]}}(D^{r})$. As $F^{[k]}$ and $\mathbb{A}_{F^{[k]}}(D^{r})$ are uncorrelated, we have that

$$Cov\left[E_{F^{[k]}}\left[\beta_{ij}^{k}\right], E_{\mathbb{A}_{F^{[k]}}(D^{r})}\left[\beta_{lp}^{k} - E\left[\beta_{lp}^{k}\right]\right]\right] = 0.$$

Hence

$$Cov\left[E_{F^{[k]}}\left[\beta_{ij}^{k}\right], E_{F^{[k+r]}}\left[\beta_{lp}^{k}\right]\right] = Var\left[H^{k}\right]_{ijlp}.$$

We can write

$$\begin{aligned} Cov\left[E_{F^{[k]}}\left[\beta_{ij}^{k}\right], E_{F^{[k+r]}}\left[\alpha_{lp}^{r}\right]\right] &= Cov\left[E_{F^{[k]}}\left[\beta_{ij}^{k}\right], E_{\mathbb{A}_{F^{[k]}}(D^{r})}\left[\alpha_{lp}^{r}\right]\right] \\ &\quad + Cov\left[E_{F^{[k]}}\left[\beta_{ij}^{k}\right], E_{F^{[k]}}\left[\alpha_{lp}^{r}\right]\right] \\ &= Cov\left[E_{F^{[k]}}\left[\beta_{ij}^{k}\right], E_{F^{[k]}}\left[\alpha_{lp}^{r}\right]\right], \end{aligned}$$

because $E_{F^{[k]}}\left[\beta^{k}\right]$ is a linear combination of $F^{[k]}$, $E_{\mathbb{A}_{F^{[k]}}(D^{r})}\left[\alpha^{r}\right]$ is a linear combination of $\mathbb{A}_{F^{[k]}}(D^{r})$, and $Cov\left[F^{[k]}, \mathbb{A}_{F^{[k]}}(D^{r})\right] = 0$. Also, because we have that $Cov\left[F^{[k]}, \alpha^{r}\right] = 0$, then $E_{F^{[k]}}\left[\alpha^{r}\right] \equiv E\left[\alpha^{r}\right]$ and

$$Cov\left[E_{F^{[k]}}\left[\beta_{ij}^{k}\right], E_{F^{[k+r]}}\left[\alpha_{lp}^{r}\right]\right] = 0.$$

Therefore,

$$Var\left[H^{[m]}\right]_{ijklpq} = \begin{cases} Var\left[H^{k}\right]_{ijlp} & q \geq k \\ Var\left[H^{q}\right]_{ijlp} & q < k \end{cases}$$

as required. □

For any β^{q}, define α^{q} such that

$$\alpha^{q} = \sum_{s=q+1}^{m} b^{s} + b^{*}.$$

By definition $Cov\left[\alpha^{q}, \beta^{q}\right] = 0$ and suppose we specify that α^{q} and $F^{[q]}$ are uncorrelated, as we did with α^{r} in the previous argument.

Proposition 5.6.2

$$Cov\left[\beta^{*}, H^{[m]}\right]_{ijlpq} = Var\left[H^{q}\right]_{ijlp}, \tag{5.20}$$

Proof: We can write

$$\begin{aligned} Cov\left[\beta^*, H^{[m]}\right]_{ijlpq} &= Cov\left[\beta^m_{ij} + b^*_{ij}, H^q_{lp}\right] \\ &= Cov\left[\beta^q_{ij} + \alpha^q_{ij}, \beta^q_{lp} - \omega^q_{lp}\right]. \end{aligned}$$

Then

$$\begin{aligned} Cov\left[\beta^*, H^{[m]}\right]_{ijlpq} &= Var\left[\beta^q\right]_{ijlp} - Cov\left[\beta^q_{ij}, \omega^q_{lp}\right] - Cov\left[\alpha^q_{ij}, \omega^q_{lp}\right] \\ &= Var\left[\beta^q\right]_{ijlp} - Var\left[\omega^q\right]_{ijlp} - Cov\left[\alpha^q_{ij}, \omega^q_{lp}\right] \\ &= Var\left[H^q\right]_{ijlp} + Cov\left[\alpha^q_{ij}, H^q_{lp}\right]. \end{aligned}$$

Writing

$$Cov\left[\alpha^q_{ij}, H^q_{lp}\right] = Cov\left[\alpha^q_{ij}, E_{F^{[q]}}\left[\beta^q_{lp}\right]\right],$$

we deduce that

$$Cov\left[\beta^*, H^{[m]}\right]_{ijlpq} = Var\left[H^q\right]_{ijlp},$$

because $E_{F^{[q]}}\left[\beta^q_{lp}\right]$ is a linear combination of $F^{[q]}$ and is, therefore, uncorrelated with α^q. □

5.6.2 An alternative model

We have presented one particular way of linking m different improved models that leads to a tractible covariance structure. This can be adjusted by observation of the adjusted coefficients of any of our improved models simply. There are many other ways we might consider specifying our beliefs about future models through structural reification. A particularly simple specification may be employed for cases where we only have one improved model that we are considering building in the future. If we let

$$\beta^1_{ij} = \rho^{'}_{ij} * \beta^0_{ij}$$

and

$$\beta^*_{ij} = \rho^*_{ij} * \beta^1_{ij},$$

then if we consider that $\rho^{'}$ and ρ^* are uncorrelated and that higher powers of these two variables are also uncorrelated, the Bayes Linear update of β^* by H^1 is tractible and easy to perform under certain conditions. We consider one such special case below.

Suppose that we can treat the matrix β^0 as known. Such an assumption may often be valid if we have run our model extensively so that we are confident that we have modelled the global behaviour well. We may have a negligible variance on the coefficients having resolved almost all of our initial uncertainty using model runs. In order to simplify the following account we also assume that $\rho^{'}$ and ρ^* are scaler random quantities. The following calculations are still valid in the more general case, however, the additional subscripts on each of the quantities appear quite cumbersome. The example we present in section 5.8 will use the case where $\rho^{'}$ and ρ^* are both scalars and so we feel it is more useful to present the calculations for this case here.

Firstly, we have that

$$E\left[\beta^1\right]_{ij} = E\left[\rho^{'}\right]\beta^0_{ij} \tag{5.21}$$

and

$$E\left[\beta^*\right]_{ij} = E\left[\rho^*\right]E\left[\rho^{'}\right]\beta^0_{ij}. \tag{5.22}$$

We also have

$$Var\left[\beta^1\right]_{ijkl} = Var\left[\rho^{'}\right]\beta^0_{ij}\beta^0_{kl}, \tag{5.23}$$

and that

$$Var\left[\beta^*\right]_{ijkl} = \left[Var\left[\rho^*\right]\left(Var\left[\rho^{'}\right] + E\left[\rho^{'}\right]^2\right) + E\left[\rho^*\right]^2 Var\left[\rho^{'}\right]\right]\beta^0_{ij}\beta^0_{kl}. \tag{5.24}$$

We then define

$$H^{'} = E_{F^1}\left[\rho^{'}\right], \tag{5.25}$$

representing the adjusted coefficients of the new model. We use relation (5.14) to link the observed adjustment with $\rho^{'}$. That is, we write

$$\rho^{'} = H^{'} + \omega^{'} \tag{5.26}$$

with $H^{'} \perp\!\!\!\perp \omega^{'}$ and $Var\left[\omega^{'}\right] = Var_{F'}\left[\rho^{'}\right]$, the amount of uncertainty left in $\rho^{'}$ after building and running the model at time t_1.

With these definitions we have

$$\begin{aligned}
E_{H'}\left[\beta^*\right]_{ij} &= E\left[\beta^*\right]_{ij} + Cov\left[\beta^*_{ij}, H'\right] Var\left[H'\right]^{-1}(H' - E\left[H'\right]) \\
&= E\left[\rho^*\right] E\left[\rho'\right] \beta^0_{ij} + E\left[\rho^*\right] \beta^0_{ij} Cov\left[\rho', \rho' - \omega'\right] Var\left[H'\right]^{-1}(H' - E\left[\rho'\right]) \\
&= E\left[\rho^*\right] \beta^0_{ij}\left[E\left[\rho'\right] + \left[\left(Var\left[\rho'\right] - Var\left[\omega'\right]\right) Var\left[H'\right]^{-1}(H' - E\left[\rho'\right])\right]\right] \\
&= E\left[\rho^*\right] \beta^0_{ij} H',
\end{aligned} \tag{5.27}$$

and

$$\begin{aligned}
Var_{H'}\left[\beta^*\right]_{ijkl} &= Cov\left[\rho^*\rho'\beta^0_{ij}, \rho^*\rho'\beta^0_{kl}\right] - Cov\left[\rho^*\rho'\beta^0_{ij}, H'\right] Var\left[H'\right]^{-1} Cov\left[H', \rho^*\rho'\beta^0_{kl}\right] \\
&= \beta^0_{ij}\beta^0_{kl}\left[Cov\left[\rho^*\rho', \rho^*\rho'\right] - E\left[\rho^*\right]^2 Var\left[H'\right]\right] \\
&= \beta^0_{ij}\beta^0_{kl}\left[E\left[\rho^{*2}\right] E\left[\rho'^2\right] - E\left[\rho^*\right]^2 (E\left[\rho'\right]^2 - Var\left[H'\right])\right] \\
&= \beta^0_{ij}\beta^0_{kl}\left[Var\left[\rho^*\right] Var\left[\rho'\right] + Var\left[\rho^*\right] E\left[\rho'\right]^2 + E\left[\rho^*\right]^2 Var\left[\omega'\right]\right].
\end{aligned} \tag{5.28}$$

The assumption that ρ' and ρ^* are uncorrelated may be quite strong in general. For certain types of analysis, however, it may be appropriate. For example, suppose the improved model that we are considering building at time t_1 is an improvement on our current model because it adds certain physics that were not previously modelled. Suppose, further, that we believe that this modelling will be thorough and that we cannot think how the reified simulator might improve upon the modelling of these new physical properties. We, instead, believe that the reified simulator would improve our modelling of the system in some other way, perhaps through careful modelling of another physical process left out of both previous models. In such cases we could assume ρ' and ρ^* were uncorrelated and would be able to use H' as a useful surrogate to F^1 for learning about $f^*(x, \theta)$. The example of structural reification and Sequential Emulation with improved models that we give in section 5.8 will use this type of relation between model coefficients.

5.6.3 Reified forecasting calculations

Having used the above objects to adjust our beliefs about β^* by $H^{[m]}$, we may use these adjusted beliefs to produce $E[y]$ and $Var[y]$ under our new beliefs about the reified simulator. We make this calculation explicit here, assuming that we are not required to adjust $\{\beta^*, u^*(x,\theta)\}$ by runs F^0. If we have runs, F^0, on $f^0(x,\theta)$, then we must update $\{\beta^*, u^*(x,\theta)\}$ by the random field induced by the collection $\{u^0(\Omega_1^0, \Theta_1^0), \ldots, u^0(\Omega_{n_0}^0, \Theta_{n_0}^0)\}$. Given relationships such as (5.12) and (5.13), this is certainly feasible. However, the extra effort involved in carrying this information up to the reified simulator only serves to complicate our exposition. To allow us to focus on the wider methodological approach for handling evolving simulators, we assume here that we have exhausted our potential for running the current model and have used all of the runs to construct an emulator with β^0 independent of $u^0(x,\theta)$, and with $u^0(x,\theta)$ being a stochastic process with known mean and variance. If we wished to include pre-updating of $\{\beta^*, u^*(x,\theta)\}$ by the random field induced by observation of F^0, the calculations would be similar in spirit to those given in section 2.3.2.

The data $H^{[m]}$, and any subsets of that array are informative for β^*. Our approach to forecasting will be to learn about the reified coefficients by adjusting our beliefs about β^* by our observations of the adjusted coefficients. We will then replace any of the moments of β^* in the forecast equations below by our adjusted moments for these quantities.

We have

$$\begin{aligned} E[y(\theta)]_i &= E[E[f^*(x^*,\theta)|x^*]]_i + \eta^*(\theta)_i \\ &= E[\beta^*]_{ij} \left[\int g(x^*,\theta)p(x^*)dx^*\right]_j + \int u_i^*(x^*,\theta)p(x^*)dx^*, \end{aligned} \tag{5.29}$$

and

$$\begin{aligned} Var[y(\theta)]_{ij} &= Var[E[f^*(x^*,\theta)|x^*]]_{ij} + E[Var[f^*(x,\theta)|x^*]]_{ij} + Var[\eta^*(\theta)]_{ij} \\ &= \int Var[f^*(x^*,\theta)]_{ij}\, p(x^*)dx^* + \int E[f^*(x^*,\theta)]_i\, E[f^*(x^*,\theta)]_j\, p(x^*)dx^* \\ &\quad - E[y(\theta)]_i\, E[y(\theta)]_j + Var[\eta^*(\theta)]_{ij}. \end{aligned}$$

This last equation may be written as

$$\begin{aligned} Var\left[y(\theta)\right]_{ij} = {} & Var\left[\beta^*\right]_{ikjp} \int g_k(x^*,\theta) g_p(x^*,\theta) p(x^*) dx^* + \int Var\left[u^*(x^*,\theta)\right]_{ij} p(x^*) dx^* \\ & + E\left[\beta^*\right]_{ik} E\left[\beta^*\right]_{jp} \int g_k(x^*,\theta) g_p(x^*,\theta) p(x^*) dx^* + Var\left[\eta^*(\theta)\right] \\ & + E\left[\beta^*\right]_{ik} \int g_k(x^*,\theta) E\left[u^*(x^*,\theta)\right]_j p(x^*) dx^* \\ & + E\left[\beta^*\right]_{jp} \int g_p(x^*,\theta) E\left[u^*(x^*,\theta)\right]_i p(x^*) dx^* - E\left[y(\theta)\right]_i E\left[y(\theta)\right]_j . \end{aligned} \tag{5.30}$$

By replacing $E\left[\beta^*\right]$ and $Var\left[\beta^*\right]$ with $E_{H^{[m]}}\left[\beta^*\right]$ and $Var_{H^{[m]}}\left[\beta^*\right]$ respectively, we are able to calculate second order moments for $y(\theta)$ under our adjusted beliefs about the reified simulator, having observed the adjusted coefficients on each of the m improved versions of the model. We write these beliefs as $E\left[y;\theta^m, H^{[m]}\right]$ and $Var\left[y;\theta^m, H^{[m]}\right]$, so that $E_{z^m}\left[y;\theta^m, H^{[m]}\right]$ and $Var_{z^m}\left[y;\theta^m, H^{[m]}\right]$ constitutes a reified decision-dependent forecast for y that has been updated by all of the observations of the system and all of the observed adjusted coefficients for each improved model, up to and including those observed at time t_m. We use $E_{z^m}\left[y;\theta^m, H^{[m]}\right]$ and $Var_{z^m}\left[y;\theta^m, H^{[m]}\right]$ as estimates to the mean and variance of $p(y|z^m, \theta^m, H^{[m]})$, and use these to characterize that distribution in the same way as demonstrated previously.

5.6.4 Decision-dependent observation forecasts

In order to use Sequential Emulation to emulate the decision tree shown in figure 5.2, our approach requires us to be able to sample from a distribution for y given everything we have observed, and from the distribution of everything we might observe at time t_k given anything observed up to that point for $k = 1, \ldots, m$. The last calculation we must describe before presenting our modified Sequential Emulation algorithm is, therefore, how we sample from distributions $p(z_{t_k}, H^k|z^{k-1}, \theta^{k-1}, H^{[k-1]})$, for $k = 1, \ldots, m$. We present here how one obtains an appropriate second order belief structure on $\{z_{t_k}, H^k\}$ that is updated by observations z^{k-1} and $H^{[k-1]}$ in a reasonable manner.

It may be that there is a natural form for the joint distribution of z_{t_k} and H^k

that can be characterized, with the aid of some expert judgements, using the second order moments that we calculate here. It is likely, however, that the objects z_{t_k} and H^k may be sufficiently dissimilar that their joint distribution is very complicated. If this is the case, a way into the problem of sampling from this distribution could involve computing the adjusted fourth order moments via the Bayes Linear variance learning methods of Chapter 8 in Goldstein and Wooff [42]. Alternatively, we may be able to marginalize the distribution into parts that may be easily sampled from. The most simple example of this would be if we were confident of the distributional form of both $p(z_{t_k}|H^k, z^{k-1}, \theta^{k-1})$ and $p(H^k|H^{[k-1]})$, we would then be able to use our methods to sample from $p(H^k|H^{[k-1]})$, and use that sample to produce a draw from $p(z_{t_k}|H^k, z^{k-1}, \theta^{k-1})$.

For reasons of time we must leave this particular problem open and we instead focus on obtaining an appropriate second order belief structure for the collection $\{z_{t_k}, H^k\}$ that is updated by z^{k-1} and $H^{[k-1]}$, for $k = 1, \ldots, m$. For the rest of this chapter we assume that, given the second order moments of $\{z_{t_k}, H^k\}$, we are able to characterize a joint distribution for these objects that we may easily sample from.

We treat the problem of establishing the required second order beliefs in a similar spirit to the way in which we have separated model and system data previously. This time, however, we must use $H^{[k-1]}$ to learn about the joint covariance of H^k and y before we adjust our beliefs about these quantities using z^{k-1}. Once we obtain the adjusted joint mean and variance of $\{H^k, y\}$ we are able to derive the adjusted moments of $\{H^k, z_{t_k}\}$ easily. Our first task then is to compute $E\left[\{H^k, y\}\right]$ and $Var\left[\{H^k, y\}\right]$ having updated our beliefs about $\{H^k, y\}$ by $H^{[k-1]}$.

We showed in section 5.6.3 how to obtain $E\left[y; \theta^m, H^{[m]}\right]$ and $Var\left[y; \theta^m, H^{[m]}\right]$, and we may obtain $E\left[y; \theta^{k-1}, H^{[k-1]}\right]$ and $Var\left[y; \theta^{k-1}, H^{[k-1]}\right]$ in precisely the same manner. Note that these last two quantities must also depend on the choice of $\theta_{t_k}, \ldots, \theta_{t_m}$, however, we are free to choose any values of these as we will only be interested in the subset $\{H^k, y_{t_0}, \ldots, y_{t_k}\}$ of $\{H^k, y_{t_0}, \ldots, y_{t_m}\}$ when calculating the moments of $\{z_{t_k}, H^k\}$, which does not depend on decisions made after time t_{k-1}. We also require $E_{H^{[k-1]}}\left[H^k\right]$, $Var_{H^{[k-1]}}\left[H^k\right]$ and an adjusted covariance between y and H^k.

Let

$$W_{ijrsv} = Cov\left[H^k, H^{[k-1]}\right]_{ijlpq} Var\left[H^{[k-1]}\right]^{-1}_{lpqrsv},$$

then we know that

$$E_{H^{[k-1]}}\left[H^k\right]_{ij} = E\left[H^k\right]_{ij} + W_{ijrsv}(H^{[k-1]} - E\left[H^{[k-1]}\right])_{rsv}$$

and

$$Var_{H^{[k-1]}}\left[H^k\right]_{ijlp} = Var\left[H^k\right]_{ijlp} + W_{ijuvw} Cov\left[H^{[k-1]}, H^k\right]_{uvwlp}.$$

Each of the quantities required in order to perform these adjustments is acquired trivially from $E\left[H^{[m]}\right]$ and $Var\left[H^{[m]}\right]$, as calculated, for example, using (5.10) and (5.19). All that remains, therefore, is to establish a covariance between y and H^k and adjust that quantity by $H^{[k-1]}$.

$$\begin{aligned} Cov\left[H^k, y\right]_{ijl} &= Cov\left[H^k_{ij}, \beta^*_{lp} g_p(x^*, \theta) + u^*_l(x^*, \theta) + \eta^*(\theta)\right] \\ &= Cov\left[H^k_{ij}, \beta^*_{lp}\right] E\left[g_p(x^*, \theta)\right] \\ &= Cov\left[H^{[m]}, \beta^*\right]_{ijklp} E\left[g_p(x^*, \theta)\right], \end{aligned} \tag{5.31}$$

which can be calculated, for example, via (5.20) and $\int g(x^*, \theta) p(x^*) dx^*$.

We know that

$$z_{t_k} = y_{t_k} + e_{t_k},$$

where e_{t_k} is independent observation error, therefore $Cov\left[H^k, z_{t_k}\right]$ is a subarray of $Cov\left[H^k, y\right]$. This is also true of $Cov\left[H^k, z^{k-1}\right]$, which implies that we can compute $E_{z^{k-1}}\left[\{H^k, z_{t_k}\}; \theta^{k-1}, H^{[k-1]}\right]$ and $Var_{z^{k-1}}\left[\{H^k, z_{t_k}\}; \theta^{k-1}, H^{[k-1]}\right]$ from $Cov\left[H^k, y\right]$, the observed system data, and the usual Bayes Linear update equations. We call the pair $E_{z^{k-1}}\left[\{H^k, z_{t_k}\}; \theta^{k-1}, H^{[k-1]}\right]$ and $Var_{z^{k-1}}\left[\{H^k, z_{t_k}\}; \theta^{k-1}, H^{[k-1]}\right]$ a decision-dependent forecast for time t_k observations z_{t_k} and H^k. More generally, adjusted moments of these types will be referred to as decision-dependent observation forecasts. If we suppose that it is possible to characterize each of our required probability distributions using decision-dependent observation forecasts, we may now sample from $p(\{H^k, z_{t_k}\}|z^{k-1}, \theta^{k-1}, H^{[k-1]})$, for $k = 1, \ldots, m$. We now write down our modified Sequential Emulation algorithm.

5.7 Sequential Emulation with evolving models

The modified Sequential Emulation algorithm that we present here follows that of section 3.6 with two key differences. Firstly, each downstream observation at time t_k, for $k = 1, \ldots, m$, is an observation of the pair $\{H^k, z_{t_k}\}$. Secondly, the distributions we must sample from in order to evaluate our expected losses are characterized by the reified decision-dependent forecasts and decision-dependent observation forecasts that were described in section 5.6. We now re-define the functions $A(\cdot)$, $\lambda_{t_k}(\cdot)$, $B^j_{\lambda^k}(\cdot)$ and $C^j_{\lambda^k}(\cdot)$ for k, $j = 1, \ldots, m$, in order to account for observations of new models.

Define

$$A(\theta^m, z^m, H^{[m]}) = \int_{-\infty}^{\infty} L(y, \theta^m) p(y|z^m, H^{[m]}, \theta^m) dy. \tag{5.32}$$

Given a second order emulator for $A(\theta^m, z^m, H^{[m]})$, we define

$$\lambda_{t_m}(\theta^{m-1}, z^m, H^{[m]}) = \arg\min_{\theta_{t_m}} \{\tilde{E}\left[A(\theta^m, z^m, H^{[m]})\right]\}. \tag{5.33}$$

Let

$$B^m_{\lambda^m}(\theta^{m-1}, z^m, H^{[m]}) = A((\theta^{m-1}, \lambda_{t_m}), z^m, H^{[m]}) \tag{5.34}$$

and define

$$C^m_{\lambda^m}(\theta^{m-1}, z^{m-1}, H^{[m-1]}) = E\left[\tilde{E}\left[B^m_{\lambda^m}(\theta^{m-1}, z^m, H^{[m]})\right]\right], \tag{5.35}$$

where the outer expectation is taken with respect to the distribution

$$p(z_{t_m}, H^m | z^{m-1}, \theta^{m-1}, H^{[m-1]}).$$

$C^m_{\lambda^m}(\theta^{m-1}, z^{m-1}, H^{[m-1]})$ is an emulator for an upper bound on our expected loss for having made decisions θ^{m-1} and having observed z^{m-1} and $H^{[m-1]}$, conditioned on time t_m strategy $\lambda_{t_m}(\theta^{m-1}, z^m, H^{[m]})$. Now, suppose we have determined strategies

$$\lambda^{k+1} = \lambda_{t_{k+1}}, \ldots, \lambda_{t_m}.$$

Let

$$B^m_{\lambda^{k+1}}(\theta^k, z^m, H^{[m]}) = \int L(y, (\theta^k, \lambda^{k+1})) p(y|z^m, H^{[m]}, \theta^k, \lambda^{k+1}) dy \tag{5.36}$$

and define

$$C^m_{\lambda^{k+1}}(\theta^k, z^{m-1}, H^{[m-1]}) = E\left[\tilde{E}\left[B^m_{\lambda^{k+1}}(\theta^k, z^m, H^{[m]})\right]\right] \tag{5.37}$$

and

$$C^{j}_{\lambda^{k+1}}(\theta^k, z^{j-1}, H^{[j-1]}) = E\left[\tilde{E}\left[C^{j+1}_{\lambda^{k+1}}(\theta^k, z^j, H^{[j]})\right]\right], \tag{5.38}$$

where the first outer expectation is taken with respect to the distribution

$$p(z_{t_m}, H^m | z^{m-1}, H^{[m-1]}, \theta^k, \lambda^{k+1})$$

and the second is taken with respect to

$$p(z_{t_j}, H^j | z^{j-1}, H^{[j-1]}, \theta^k, \lambda^{k+1})$$

for $j = k+1, \ldots, m-1$. Then, we can define

$$\lambda_{t_k}(\theta^{k-1}, z^k, H^{[k]}) = \arg\min_{\theta_{t_k}}\{\tilde{E}\left[C^{k+1}_{\lambda^{k+1}}(\theta^k, z^k, H^{[k]})\right]\} \tag{5.39}$$

for $k = 1, \ldots, m-1$. These definitions are the same, in essence, to those given in section 3.5. This is because we simply treat each observation at time t_k as the pair $\{H^k, z_{t_k}\}$, instead of just a z_{t_k} as we had before. The only reason for not re-defining z_{t_k} to be both system and model observations made at time t_k is because of the way each observation is used to characterize the relevant distribution via the two-stage update presented in section 5.6.

5.7.1 Reified Sequential Emulation algorithm

We now present the Reified Sequential Emulation algorithm. As with the Sequential Emulation algorithm presented in section 3.6, we begin the algorithm with a set of preliminary steps designed to determine our current beliefs about the models and system, and to determine how those beliefs will be used to characterize the various distributions required for Sequential Emulation. One of the key differences between the two algorithms is that we make no provision here for the use of fully probabilistic model and system beliefs. The methods of forecasting via the reified form that we have presented in this chapter use Bayes Linear methods to update second order beliefs about the quantities of interest. Extending the methods of reified forecasting to include fully probabilistic descriptions of our models and the relationships between them is beyond the scope of this thesis.

Preliminaries

We begin with a computer simulator for the complex system, $f^0(x, \theta)$, and know that at each of our intervention points, $t_1, \ldots, t_m$, we have the option of building and making runs on an improved version of the simulator $f^k(x, \theta)$, where the decision of whether or not to build each simulator and how much we are to run it if built is part of θ. We assume that we have exhausted our ability to run $f^0(x, \theta)$ and that we have used the runs we have already to build an emulator for $f^0(x, \theta)$ in the form of equation (5.7), with β^0_{ij} and $u^0_i(x, \theta)$ independent and with $E\left[\{\beta^0_{ij}, u^0_i(x, \theta)\}\right]$ and $Var\left[\{\beta^0_{ij}, u^0_i(x, \theta)\}\right]$ specified for all i and j. For discussion of this assumption refer back to section 5.6.3.

We assume that our emulator for any simulator is of the form given in equation (5.8).

RP1

Specify a second order belief structure over each of the $m + 1$ simulators and the reified simulator. As we have established a form for any emulator for each of the models through (5.8), this structure should be specified in terms of the coefficient matrices $\{\beta^0, \ldots, \beta^m, \beta^*\}$ and the residual processes $\{u^0(x, \theta), \ldots, u^*(x, \theta)\}$. We provided an example of such a structure with (5.10), (5.11), (5.12), and (5.13).

RP2

Specify beliefs about x^*, $\eta^*(\theta)$ and e. We require a prior probability distribution for x^* and second order moments for both $\eta^*(\theta)$ and e.

RP3

Select both 'accurate' and 'fast' forecasting methods. This is the same as for P3 in the original Sequential Emulation algorithm, with the exception that the forecast integrals are those presented in section 5.6.3 and involve components of the reified model emulator.

RP4

Decide how forecasts will be used to sample from the required distributions. We must decide here how the reified decision-dependent forecasts

$$E_{z^m}\left[y;\theta^m,H^{[m]}\right], \qquad Var_{z^m}\left[y;\theta^m,H^{[m]}\right],$$

and the decision-dependent observation forecasts

$$E_{z^{k-1}}\left[\{H^k,z_{t_k}\};\theta^{k-1},H^{[k-1]}\right], \qquad Var_{z^{k-1}}\left[\{H^k,z_{t_k}\};\theta^{k-1},H^{[k-1]}\right]$$

for $k=1,\ldots,m$, are to be used to characterize the probability distributions on our decision tree.

RP5

Specify numerical integration methods. This is the same as in step P5 in the original Sequential Emulation algorithm.

The preliminary steps RP1 to RP5 allow us to obtain 'coarse' and 'accurate' samples from our required distributions. We now use this ability in the Reified Sequential Emulation algorithm.

RS1

Construct a multi-level emulator for $A(\theta^m,z^m,H^{[m]})$.

RS2

Define $\lambda_{t_m}(\theta^{m-1},z^m,H^{[m]})$ **as in (5.33)**.

RS3

Construct a multi-level emulator for $B^m_{\lambda^m}(\theta^{m-1},z^m,H^{[m]})$. This follows the same process as for **RS1**, except we set

$$\theta^m=(\theta^{m-1},\lambda_{t_m}(\theta^{m-1},z^m,H^{[m]})).$$

RS4

Construct a multi-level emulator for $C^m_{\lambda^m}(\theta^{m-1},z^{m-1},H^{[m-1]})$**.**

RS5

If $m = 1$ proceed to *RS11*. Else set $k = m - 1$ and go to *RS6*.

RS6

Define $\lambda_{t_k}(\theta^{k-1}, z^k, H^{[k]})$ using the emulator for $C^{k+1}_{\lambda^{k+1}}(\theta^k, z^k, H^{[k]})$ and (5.39).

RS7

Construct a multi-level emulator for $B^m_{\lambda^k}(\theta^{k-1}, z^m, H^{[m]})$ as defined in (5.36).

RS8

Construct a multi-level emulator for $C^m_{\lambda^k}(\theta^{k-1}, z^{m-1}, H^{[m-1]})$ as defined in (5.37). Set $j = m - 1$.

RS9

Construct a multi-level emulator for $C^j_{\lambda^k}(\theta^{k-1}, z^{j-1}, H^{[j-1]})$. Let $j = j - 1$.

RS10

If $j \geq k$ go back to RS9. Else if $k = 1$ proceed to *RS11*. Else set $k = k - 1$ and go back to *RS6*.

RS11

Stop.

5.8 Illustrative example

We present here a relatively short example that applies the methods introduced in this chapter to the CO_2 abatement problem of chapter 4. We intend to give a simplified structural reification of C-GOLDSTEIN and use Reified Sequential Emulation to plot an emulator of an expected upper bound on our expected loss for making policy today. Although all of the decision support tools we discussed in chapter 4,

such as pruning and risk profiling, are available to use once the Reified Sequential Emulation has been performed, we do not apply these tools in this example. Our goal is to provide a concrete example to reinforce the theory we have presented and to show some of the extra problems that may be explored within this framework.

The main new area of decision support we are able to offer involves the question of whether or not to build, and how much to run, an improved simulator. As has been discussed, and as we will show in our example, these decisions are directly included in the Reified Sequential Emulation. The framework that we have established allows us to provide decision support for these questions. So, for example, we can explore how the amount of perceived improvement of our model can affect our choices regarding the building and running of the improved simulator as well as the way in which it affects our abatement policy strategy.

Although this example is a useful illustration of how our methodology works in practice, we must give the same caveats as we did in chapter 4. Namely, that a number of the simplifications we will make in order to perform the Reified Sequential Emulation quickly would not be acceptable in a case study or a genuine attempt at providing decision support. We discuss this further below.

5.8.1 Structural reification of C-GOLDSTEIN

We have the model C-GOLDSTEIN, described fully in section 4.2, which we label $f^0(x, \theta)$. The CO_2 abatement problem is set up as before, where we imagine that we must make an abatement policy for 1995 that may be altered in 2035, following an observation of the global mean temperature anomaly. We also imagine that in 2035 we may have access to an improved version of C-GOLDSTEIN that has the same five inputs we fixed in chapter 4 and which we label $f^{'}(x, \theta)$.

In order to link these two models to each other and to the system, we introduce the reified simulator, $f^*(x, \theta)$, that links to the global temperature anomaly, y, via (5.6), and that also has the same five inputs we worked with previously. In order to perform Reified Sequential Emulation for this problem we begin at RP1 by linking our three simulators via a second order belief structure.

We assume that each model satisfies (5.8) with $g(x, \theta)$ as on page 103. To simplify

our description we use the alternative model introduced in section 5.6.2 and let

$$\beta_{ij}^{'} = \rho^{'}\beta_{ij}$$

and

$$\beta_{ij}^{*} = \rho^{*}\beta_{ij}^{'}$$

with $\rho^{'}$ and ρ^{*}, and $\rho^{'2}$ and ρ^{*2}, treated as uncorrelated scalar random quantities. As we discussed at the end of section 5.6.2, this might imply that our improved simulator models a physical process not included on C-GOLDSTEIN, and that we expect this modelling to be thorough enough so that we cannot think of any way in which the reified model might improve upon the modelling of this particular new process, but would improve our model in other ways. We link the residual processes via

$$u^{'}(x,\theta) = u(x,\theta) + \delta^{'}(x,\theta)$$

and

$$u^{*}(x,\theta) = u^{'}(x,\theta) + \delta^{*}(x,\theta),$$

with $u(x,\theta) \perp\!\!\!\perp \delta^{'}(x,\theta)$, $u^{'}(x,\theta) \perp\!\!\!\perp \delta^{*}(x,\theta)$, and $\delta^{'}(x,\theta) \perp\!\!\!\perp \delta^{*}(x,\theta)$. We assume that both $\delta^{'}$ and δ^{*} are mean-zero random processes.

To simplify the problem further still, we assume that the β_{ij}s are known and that $u(x,\theta)$ has given mean and variance. This means that the second order moments of $\beta^{'}$ and β^{*} are calculated via (5.21), (5.22), (5.23), and (5.24). Our beliefs about the residual processes across models are

$$E\left[u^{*}(x,\theta)\right] = E\left[u^{'}(x,\theta)\right] = E\left[u(x,\theta)\right]$$

and

$$Var\left[u^{'}(x,\theta)\right] = Var\left[u(x,\theta)\right] + Var\left[\delta^{'}(x,\theta)\right]$$
$$Var\left[u^{*}(x,\theta)\right] = Var\left[u^{'}(x,\theta)\right] + Var\left[\delta^{*}(x,\theta)\right].$$

All that remains in order to link the models and complete RP1, then, is to specify β_{ij} for $i = 1,2,3$, $j = 1,\ldots,8$ and the beliefs $E\left[\rho^{'}\right]$, $E\left[\rho^{*}\right]$, $Var\left[\rho^{'}\right]$, $Var\left[\rho^{*}\right]$, $E\left[u(x,\theta)\right]$, $Var\left[u(x,\theta)\right]$, $Var\left[\delta^{'}(x,\theta)\right]$ and $Var\left[\delta^{*}(x,\theta)\right]$. First we set

$$E\left[\rho^{'}\right] = E\left[\rho^{*}\right] = 1.$$

We have chosen to treat the matrix β as known in order to simplify our calculations, therefore, a natural choice for this matrix is $E_F[\beta]$, calculated using the 100 runs we adjusted on in chapter 4. We set $\beta_{ij} = E_F[\beta]_{ij}$, and allow the emulator we use for C-GOLDSTEIN in this chapter to have the same expectation as our original emulator by setting $u(x,\theta)$ to be the random field generated by adjusting the prior beliefs given in chapter 4 by F. This specification is equivalent to allowing our beliefs about the random field $u(x,\theta)$ to be the same as with our adjusted emulator of chapter 4, but ignoring the covariance between β and $u(x,\theta)$ induced by F to simplify the calculations.

This alternative specification of an emulator for C-GOLDSTEIN has less variance than in our example of chapter 4 as both the contribution of $Var\left[\beta^0_{ij} g_j(x,\theta)\right]$ and $Cov\left[\beta^0_{ij}, u^0_k(x,\theta)\right]$ are ignored. The results of both analyses will, therefore, not be strictly compatible. However, as our goal is to illustrate the methods and not to solve the CO_2 abatement policy problem for real, we feel that these simplifications are beneficial.

The remaining values that we must specify, namely $Var\left[\rho'\right]$, $Var\left[\rho^*\right]$, $Var\left[\delta'(x,\theta)\right]$ and $Var\left[\delta^*(x,\theta)\right]$, will define how much uncertainty about the temperature anomaly we feel we can resolve by building better models, the proportion of that uncertainty that can be resolved by the model we might build in 2035, and the amount of uncertainty we may resolve by observing emulator coefficients only. Given the values of the other quantities that we have specified, we determine the distance between $f'(x,\theta)$ and $f(x,\theta)$ through $Var\left[\rho'\right]$ and $Var\left[\delta'(x,\theta)\right]$, and the distance between $f^*(x,\theta)$ and $f'(x,\theta)$ through $Var\left[\rho^*\right]$ and $Var\left[\delta^*(x,\theta)\right]$. The nature of our methods mean that we only learn about $Var\left[\rho^*\right]$ by choosing to build and run the new model. Therefore, if the majority of our variance on $f'(x,\theta)$ is located in $Var\left[\delta'(x,\theta)\right]$, we cannot hope to change our beliefs today about the system much by building $f'(x,\theta)$.

Another thing to consider when specifying these quantities is that, in some sense, we are bounded in our choices by the overall variance for y. In fact, were it our desire for this example to be compatible with the analysis of chapter 4, we would require that $Var\left[y(\theta)\right]$, as calculated when producing forecasts in chapter 4, was exactly equal to $Var\left[f^*(x,\theta)\right] + Var\left[\eta^*(\theta)\right]$. Although we are not aiming for this

compatibility, we would like our example to be sensible and must choose the required variances and the variance on discrepancy $Var\left[\eta^*(\theta)\right]$ so that the values of $Var\left[y(\theta)\right]$ computed in both examples are not totally dissimilar. It is worth commenting that this type analysis might cause us to revise our overall judgements about the discrepancy of C-GOLDSTEIN, and therefore revisit the analysis in chapter 4.

Both $Var\left[\delta^{'}(x,\theta)\right]$ and $Var\left[\delta^*(x,\theta)\right]$ are prior variances for a process that we will not observe. We treat both as having components that are independent across time so that we must specify a 3×3 diagonal matrix for each. We choose both so that they are each one tenth the standard deviation specified for $u(x,\theta)$ in chapter 4. Relative to the variance of the coefficients, this has the effect of maintaining only a small contribution of these unseen processes. Hence we specify that there can be a significant change in our uncertainty about the system brought about by building and running $f^{'}(x,\theta)$.

We choose $Var\left[\eta^*\right]$ directly by keeping its correlation matrix the same as that for η in chapter 4, but dividing the standard deviations by 10. This means that we believe there is very little discrepancy between the reified model and actual climate. The key choice, in the context of this example, is in $Var\left[\rho^{'}\right]$ and $Var\left[\rho^*\right]$ but, more specifically, in the difference between the two. Our intention is to choose combinations of these two quantities that lead to roughly the same prior variance for $y(\theta)$, but that determine different relative distances between the simulators. Our first choice is $Var\left[\rho^{'}\right] = 0.25$ and $Var\left[\rho^*\right] = 0.1$. This means that $Var\left[f^*(x,\theta)\right]$ is much closer to $Var\left[f^{'}(x,\theta)\right]$ than $Var\left[f^{'}(x,\theta)\right]$ is to $Var\left[f(x,\theta)\right]$. This implies that resolving all of the uncertainty on the coefficients for $f^{'}(x,\theta)$ will tell us a lot about $f^*(x,\theta)$ and, therefore, about $y(\theta)$. We shall discuss this choice and compare results with an alternative parametrization after demonstrating the rest of the analysis.

5.8.2 Decisions regarding the improved model

In 2035 we observe $H^{'} = E_{F^{'}}\left[\rho^{'}\right]$, the adjusted coefficients of the new model. We use relation (5.14) to link the observed adjustment with $\rho^{'}$. That is, we write

$$\rho^{'} = H^{'} + \omega^{'}$$

with $H' \perp\!\!\!\perp \omega'$ and $Var\left[\omega'\right] = Var_{F'}\left[\rho'\right]$, the amount of uncertainty left in ρ' after building and running the model. $Var\left[\omega'\right]$, hereafter κ, is a decision parameter and is part of θ_{t_0}. Having chosen $Var\left[\rho'\right] = 0.25$, we know that $\kappa \in [0, 0.25]$, where $\kappa = 0.25$ corresponds to not building $f'(x, \theta)$ at all.

In order to treat this decision properly, there must be a cost associated with building the model and one associated with performing runs and resolving uncertainty on the coefficients once the model is built. Furthermore, these costs must be added to our loss function DICE in a way that is intuitive. The most natural way of incorporating model building and running costs in DICE is to add to the investment function $I(t)$ at 2035 (for full details regarding the investment function and how it is incorporated into DICE see appendix C).

For $t = 2035$ we can compute $I(t)$ as a function of global output in the usual way and then add a separate model investment. This will be in the form of a one-off building cost, B, and a run cost that is a function of the proportion of uncertainty resolved. Let

$$\alpha = \frac{\kappa}{Var\left[\rho'\right]}, \qquad \hat{\alpha} = \max(\alpha, M_\alpha)$$

where M_α is chosen so that $\log(M_\alpha)$ is large, but not infinite as it would be at 0. Then at $t = 2035$, having computed the normal $I(t)$, we let

$$I(t) = I(t) + B\mathbb{1}_{\hat{\alpha}<1} - K\log(\hat{\alpha}),$$

where $\mathbb{1}_{\hat{\alpha}<1}$ is the indicator function that is equal to 1 if $\hat{\alpha} < 1$ and 0 otherwise.

Here $K \log M_\alpha$ is set to be the cost of running the model at every location. We choose B to be \$100 million and K such that the cost of running the model everywhere was approximately \$100 million again. The build cost of \$100 million would be made up of the cost of performing the research as well as the coding and building of any new facilities required in order to run the model. These costs may seem large, however on DICE's scale they are tiny. To put these costs into perspective, total consumption over the ten decade period is of the order of \$470 trillion.

5.8.3 Completing the preliminaries

Before beginning our Reified Sequential Emulation we must complete steps RP2-RP5. Beginning at RP2, we have already specified our discrepancy η^* and keep our distribution on x^* and our beliefs about e the same as the example in chapter 4. Step RP3 requires us to determine our methods of coarse and accurate forecasting and, because we use the same vector of regressors, the same correlation function for $u(x, \theta)$, and the same uniform distribution on x^*, all of the forecast integrals in (5.29) and (5.30) can be computed analytically as we did for the forecast integrals in chapter 4.

Step RP4 requires us to choose a way of characterizing and sampling from the required distributions using our reified decision-dependent forecasts and our decision-dependent observation forecasts. Addressing the former first, we may use the same multivariate log-normal distributional assumption for y as was used in chapter 4. The latter types of forecast give joint second order beliefs on $\{z_{t_1}, H^{'}\}$. The observation $H^{'}$ is an indicator of the value of $\rho^{'}$, which is a scalar multiplier of our regression surface for C-GOLDSTEIN. If $\rho^{'}$ were negative, that would correspond to a model that had the opposite behaviour to C-GOLDSTEIN. One implication of this is that the new model would determine that global temperature will decrease as the amount of CO_2 emitted is increased. We may, therefore, consider $H^{'}$ to be strictly positive and use a joint log-normal distributional assumption on $\{z_{t_1}, H^{'}\}$ for convenience.

This particular choice, whilst computationally convenient, does leave a problem. Some exploratory analysis showed that high values of $H^{'}$ were occasionally sampled, which made the forecasts non-physical. For example, a sampled value of $H^{'}$ of 5 could mean that we expect future temperature to be 15° higher than today according to our forecast. To actually expect such a value seems ludicrous, therefore, our solution is to restrict the range of possible $H^{'}$ to be

$$H^{'} \in \left[0, 1 + \max\left(4 * SD_{z_{t_0}}[H^{'}]\right)\right]$$

which, for this example, gives $H^{'} \in [0, 2.28]$. Testing of this assumption showed that the largest amount of probability ignored by choosing $H^{'}$ in this way was 0.003. We

felt that for the purposes of providing an illustration that was computationally efficient, this choice was pragmatic and would not affect our conclusions.

The numerical integration methods we are required to specify in RP5 will be the same as those used in our previous example.

5.8.4 Calculations

We must be able to derive the moments of $y(\theta)$, both before and after seeing a given value of H^1, in order to be able to compute the two types of forecast that we require. To do this we use equations (5.29), (5.30) and (5.31), each of which is a function involving $E[\beta^*]$ and $Var[\beta^*]$. In order to update our beliefs about the system using the observed expected coefficients, $H^{'}$, we adjust the moments of β^* by this quantity using equations (5.27) and (5.28), before computing the moments of $y(\theta)$ in the normal way.

We now perform the Reified Sequential Emulation algorithm by building the emulators of $A(\theta^1, z^1, H^{'})$, $B^1_{\lambda^1}(\theta_{t_0}, z^1, H^{'})$ and $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$ in sequence. The methods and calculations undertaken were almost exactly the same as those in section 4.6 and, as such, we do not describe the method for building each emulator exactly here. Instead, we comment on the differences between our approaches and comment on some of the features we identified whilst building emulators in this example.

The first minor difference is that we used Sobol sequences (introduced in section 2.4.1) instead of Latin Hypercubes to generate designs for coarse and accurate evaluations of both emulators to enable us to obtain the higher dimensional designs required for each set of runs more quickly. Another difference between the two examples is that we had to decide how the maximum of κ would be handled.

Recall that if $\kappa \in [a, b]$, then $\kappa = b$ corresponded to the decision not to build the new model at all. However, when designing runs and building emulators it is possible that we may try evaluating one of our emulators at $\kappa = b$ with $H^{'} \neq 1$ by mistake. This is not feasible since, if we do not build the new model, we cannot observe its coefficients. Therefore, when writing the computer code to perform this Sequential Emulation we provide a number of safety nets to ensure that this does not happen. Firstly, if we wish to compute a reified decision-dependent forecast with

$\kappa = b$, then we always use the prior, unadjusted, moments of β^* in our calculations. Secondly, we restrict our designs so that those to be used for building emulators for $A(\theta^1, z^1, H')$ and $B^1_{\lambda^1}(\theta_{t_0}, z^1, H')$ always have a small subset with $H' = 1$, $\kappa = b$ and a Sobol sequence in the remaining inputs. Always including this subset means that we are able to specifically explore our utility for not paying B and not building the new model.

We use precisely the same methods for building our multi-level emulators as we used in section 4.6. We use the same heuristics and techniques for choosing the required parameters and, indeed, the only difference between the modelling in the two examples is the two extra inputs we must deal with. We introduce no special treatment for the 5-dimensional and 4-dimensional multi-level emulations required for emulating $A(\theta^1, z^1, H')$ and $B^1_{\lambda^1}(\theta_{t_0}, z^1, H')$, other than having to fit extra regression terms in our saturated linear models, choose extra correlation parameters using our heuristic, and observe more residual plots in order to aid model selection.

We do not present a detailed account of our step by step Sequential Emulation in this example as we would essentially be repeating ourselves. In chapter 4, we fully demonstrated the techniques of building multi-level emulators in sequence and to do so again here seems unnecessary. We do, however, present selected details of the built emulators in appendix E.4. For the same reason, we do not perform the detailed diagnostics, analysis of strategy plots, risk profiling and pruning that we performed in chapter 4. This is because, in spirit at least, these would be the same in this example. If we were actually providing decision support for this problem and accounting for the possibility of building and running $f'(x, \theta)$, then we would spend a large proportion of our time using the policy support techniques described in section 3.7 and demonstrated in section 4.7, as this kind of policy support is our ultimate goal. The goal achieved by this example is to illustrate the new ideas presented in this chapter, specific to the introduction of observations of new, improved, simulators in the future. Our other goal in presenting this example is to show some of the new areas of decision support that are open to us when considering the impact of future simulators; this we consider now.

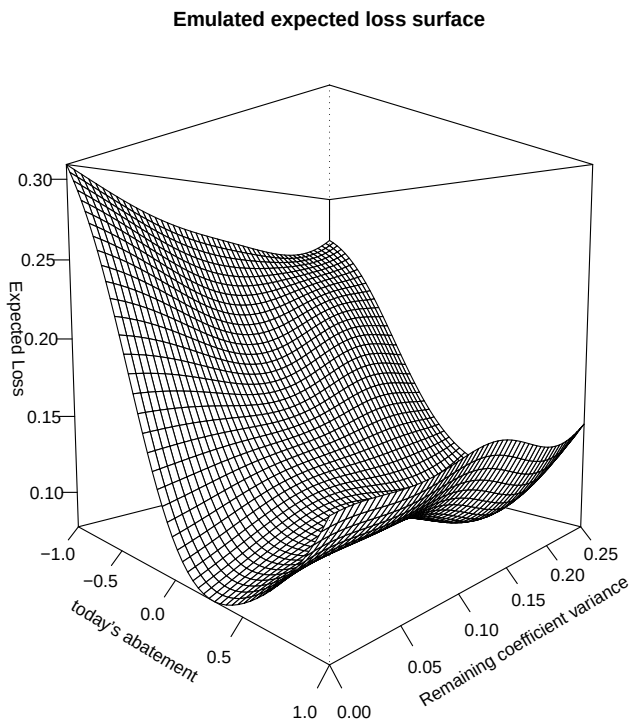

Figure 5.3: A plot of our emulated upper bound $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ where $Var\left[\rho'\right] = 0.25$ and $Var\left[\rho^*\right] = 0.1$.

5.8.5 Additional policy support

We present our emulator for the emulated upper bound $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ in this example in figure 5.3. This surface is both a function of θ_{t_0} and κ and shows some interesting behaviour. The first thing to notice is that the value of θ_{t_0}, our level of CO_2 abatement, is still the dominating influence on our expected loss. What is more, the shape of this curve and, indeed, the location of the 'optimal' decision, is very similar to that of our original example. This causes us to suspect immediately that some kind of action against rising CO_2 emissions should be taken now, regardless of the fact that our understanding of climate behaviour may be more advanced in the future. This is not surprising as the original parametrzation of DICE had been chosen so as to penalize deferred abatement.

We also notice that we do better, from an expected loss perspective, by building the model and resolving some of the uncertainty in the coefficients, than by not

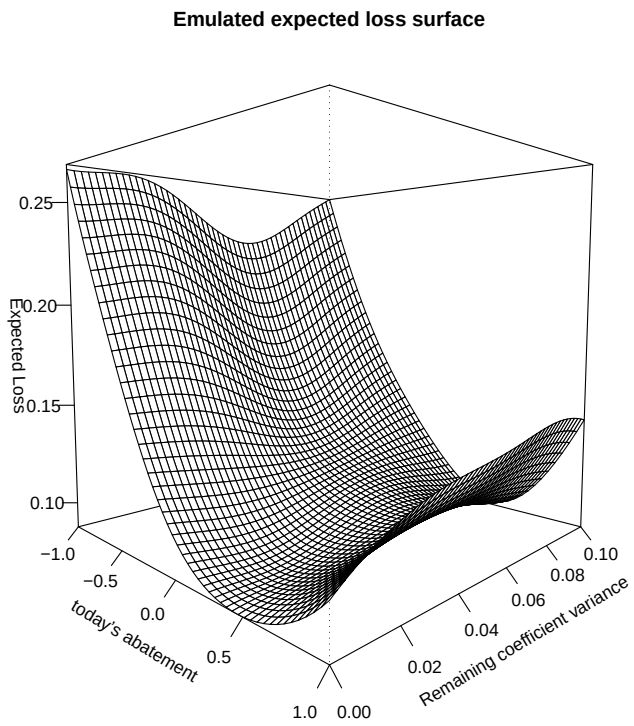

Figure 5.4: A plot of our emulated upper bound $\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]$ where $Var\left[\rho^{'}\right] = 0.1$ and $Var\left[\rho^*\right] = 0.25$.

building the model at all for any value of θ_{t_0}. If we were to use this emulator as a surrogate for our expected loss surface, then the optimal decision is to set $\kappa = 0$ and to choose $\theta_{t_0} \approx 0.68$, which corresponds to an abatement of 32% of BAU emissions per decade, and to building the model and resolving all of the uncertainty in the coefficients.

We now repeat our analysis with an alternative distribution of the total variance in the system. By letting $Var\left[\rho^{'}\right] = 0.1$ and $Var\left[\rho^*\right] = 0.25$ we keep the same overall variance for y, but are stating that the model we might build in the future will not resolve as much of our uncertainty about y as our previous parametrization allowed. We keep all costs for building and resolving uncertainty in the model the same and perform a new Reified Sequential Emulation with this alternative configuration of the variances of the improved and reified simulators.

The results from this Sequential Emulation are plotted in figure 5.4. What we

see from this diagram, as well as the qualitative behaviour of our loss surface being similar in θ_{t_0}, is that not building the model at all is better than building it and not resolving most of the uncertainty. For instance, if there were insufficient time to run the model everywhere between now and 2035 then it may be better not to build the improved simulator at all. It also suggests that there may be some threshold for $Var\left[\rho^{'}\right]$ below which we should not build the simulator. This, in turn, suggests that these methods may be used to give insight into the important question "how good does my simulator have to be to be worth building?" We discuss this idea further in section 5.9.

We are also interested in the comparison between these two analyses. The 'optimal' decision for this parametrization was $\theta_{t_0} \approx 0.76$ and $\kappa = 0$, suggesting again that we should build the new model and resolve all of our uncertainty about the coefficients. We found the minimum expected loss for both of our parametrizations and computed the difference between them. The analysis for $Var\left[\rho^{'}\right] = 0.25$ had minimum expected loss around 0.01 higher than that for the alternative parametrization. As we argued at the end of chapter 4, this corresponds to roughly \$20 billion dollars saved per decade if the model we might build resolves most of the variance in the system, as it does in this example, compared with the case where it does not.

It is worth highlighting that these conclusions are only valid if we are confident in our Sequential Emulation of the loss surface. In a case study, or more careful analysis it would be essential that pruning exercises and diagnostic testing of our emulators, such as via the forward sampling methods discussed in chapter 4, take place before we draw any conclusions for the shape of our loss surface. These diagrams and conclusions are given as examples of the kinds of decision support we may be able to provide with a careful application of this new version of the methodology.

5.9 Discussion

5.9.1 Feasibility of Reified Sequential Emulation

There are two main issues regarding the feasibility of the methods described in this chapter. The first is the issue of specifying beliefs jointly across all future

simulators, including the imaginary reified simulator. Whilst this specification will be very challenging, the issue of feasibility of the reifying technology is discussed in detail elsewhere (see, for example, Goldstein and Rougier [40], particularly in the rejoinder of the following discussion) and is beyond the scope of this thesis. We view the reification technology here as a pragmatic way of linking a collection of models in order to facilitate the types of decision support we have described. The second issue concerns the feasibility of the Reified Sequential Emulation itself, once our prior beliefs have been written down. In section 3.8 we discussed the feasibility of performing the Sequential Emulation algorithm that did not account for observations on improved models.

The extra difficulty in performing the Reified Sequential Emulation algorithm, once the preliminary steps have been completed, corresponds to an increase in dimensionality caused by observing coefficients on future models as well as data from the complex system at each time step. Suppose, for example, that each β^k is a matrix with M_β elements. Due to the way we have defined $g(x, \theta)$, many of these elements will be identically equal to zero. However, when sampling from some distribution $p(z_{t_k}, H^k | z^{k-1}, \theta^{k-1}, H^{[k-1]})$, we are sampling from a distribution with maximum dimension $dim(z_{t_k}) + M_\beta$. Even if many of the elements of β^k are identically zero, we may still be sampling from a high dimensional distribution.

The arguments for the feasibility of generating enough samples from our distributions in order to be able to build coarse and accurate emulators at each stage of the algorithm do not differ from those given in section 3.8. Namely, that feasibility depends on the availability of resources, computing power, and, in extreme cases, may require additional techniques to be developed. The examples we gave included exploiting the structure of our emulators to make parts of our loss integrals analytic. Another idea, specific to the integrals we must solve in a Reified Sequential Emulation, would involve dimension reduction, at least for the coarse evaluations. We could achieve this by only allowing a smaller subset of the model coefficients, one that we judge to be the most influential for our expected loss, to differ when sampling from $p(z_{t_k}, H^k | z^{k-1}, \theta^{k-1}, H^{[k-1]})$.

Another computational challenge particular to this adaptation of our method-

ology will be the large increase in the number of variables we must emulate on. Consider, for example, our emulator for $A(\theta^m, z^m, H^{[m]})$. This is a function with up to mM_β additional variables compared with the function $A(\theta^m, z^m)$ that we defined in our previous version of the algorithm. The methods of active variables, discussed in section 2.2.6, will play an important role in reducing the number of runs required to build these emulators and making the problem feasible.

Although the computational challenges that a real-world application of this methodology would present may seem arduous, it would be worth overcoming them if only to learn the value of building an improved model. Taken in the context of the costs involved in the policy decisions that must be made and the amount of money invested in computer models, the challenges presented by large problems should be addressed rather than shied away from.

5.9.2 Further policy support

The idea, presented in our example, of comparing our optimal policies when faced with two different improved simulators, each potentially resolving different amounts of uncertainty about the system, can be generalized and used to provide insight into other questions. We could, for example, compare our results for a number of different potential levels of improvement to give insight into the question of how good an improved simulator must be in order for it to be worth building.

We could, in theory, go further and, for example, emulate the optimal policy as a function of the uncertainty in the coefficients on the improved model. Another idea we might consider is to introduce a formal uncertainty on the quantity $Var\left[\beta^k\right]$ for $k = 1, \ldots, m$, and to investigate the optimal policy and the decision of whether or not to build and how much to run each model in the case when the quality of each model that we might build is not known for certain. These are both potential areas for further work.

Another question we might provide insight into within this framework is: "how much would the new model have to cost before building it became non-optimal?". This is similar, in spirit, to the ideas discussed in section 3.8 regarding treating the loss function as a computer model about whose inputs and outputs we are uncertain.

Although we may not have to introduce an uncertainty on build cost in order to answer this particular question, we may attempt to emulate our decision regarding the building and running of the new simulator as a function of the build cost. We might then treat the search for a threshold cost as an optimization problem. These ideas are possible areas for further work.

Chapter 6

Bayes Linear calibrated decision-dependent forecasts

In this chapter we return to the case where we have the same model now and in the future, in order to add a calibration step to our Bayes Linear decision-dependent forecasts. In section 6.1 we discuss current methods of Bayes Linear calibrated forecasting for models without decision inputs, and describe a method of obtaining the required quantities for this technique via a large-scale sampling experiment. In section 6.2 we generalize these methods to models with decision inputs and describe how we may exploit the spin up property that many of these models have. We define a quantity called the *hat function* and describe how this quantity may be used to provide Bayes Linear calibrated decision-dependent forecasts. We argue that the sampling methods that are appropriate when we only have a single hat-run, will often be inappropriate when working with a hat function. We describe how a computer algebra package may be employed, together with sampling, to emulate the hat function moments in an appropriate manner.

In section 6.3 we illustrate how our Sequential Emulation algorithm may be adapted to include Bayes Linear calibrated forecasts based on system observations we have now, if required. We conclude the chapter by offering some early ideas regarding how the algorithm might be adapted if we allowed future system observations to define new hat functions. This last problem is left open.

6.1 The Hat run

In section 2.3.2, we made a case for ignoring the information contained in observations z_{t_0} about the best input x^* when computing Bayes Linear forecasts. By first integrating x^* out of (2.2) to derive moments for $y(\theta)$ and then adjusting these moments by z_{t_0}, we effectively ignored this information. We argued that this may often be appropriate, stating that the effort involved in its proper handling may be a waste of time and resources, particularly in the case where we have already performed a history match for the computer model using z_{t_0}, and when we have obtained good global fits to our model data via $\beta g(x, \theta)$. We also stated that if further runs of the computer model were available to us, inserting a calibration step and obtaining model runs at the expected value of x^* could yield more powerful forecasts. We shall describe these techniques here.

We suppress any dependence of the model f on decisions θ for the moment, and consider the simulator with only model inputs, $f(x)$. The paper by Goldstein and Rougier [39] describes a method of Bayes Linear calibrated forecasting via a quantity they call the *hat run*. In order to put the information contained in z_{t_0} regarding x^* into the tractible Bayes Linear forecasting methods, they adjust beliefs about x^* by the system data z_{t_0}, and then run the computer model at the adjusted expectation of x^*. Beliefs about y may then be adjusted by both z_{t_0} and this new model run.

They define

$$\hat{x} = E_{z_{t_0}}[x^*] = E[x^*] + Cov[x^*, z_{t_0}]\, Var[z_{t_0}]^{-1} (z_{t_0} - E[z_{t_0}]), \tag{6.1}$$

and then the hat run is defined to be

$$\hat{f} = f(\hat{x}).$$

The Bayes Linear calibrated forecast is then

$$E_{z_{t_0},\hat{f}}[y] = E[y] + Cov\left[y, (z_{t_0}, \hat{f})\right] Var\left[z_{t_0}, \hat{f}\right]^{-1} \left((z_{t_0}, \hat{f}) - E\left[(z_{t_0}, \hat{f})\right]\right) \tag{6.2}$$

and

$$Var_{z_{t_0},\hat{f}}[y] = Var[y] - Cov\left[y, (z_{t_0}, \hat{f})\right] Var\left[z_{t_0}, \hat{f}\right]^{-1} Cov\left[(z_{t_0}, \hat{f}), y\right]. \tag{6.3}$$

In principle then, we can include the information about x^* contained in z_{t_0} as part of a forecast that retains the tractibility of the Bayes Linear forecasting methodology.

The rationale for the hat run method is as follows. The model runs, F, inform us about the global behaviour of our models and help us to fix the surface $\beta g(x)$ and many of the global characteristics of the surface $u(x)$. We use these features to derive our beliefs about y, and then observations z_{t_0} adjust these global beliefs. Suppose, for example, that we have two time points t_0 and t_1. The observation z_{t_0} is informative for y_{t_1} through the link between the discrepancies and between $f_{t_0}(x^*)$ and $f_{t_1}(x^*)$. If we imagine that we have captured the global behaviour in the model well, then observing the global behaviour in $f_{t_0}(x^*)$ will be informative for the global behaviour in $f_{t_1}(x^*)$. However, suppose that there is very little correlation between $u_{t_0}(x)$ and $u_{t_1}(x)$. Then adjusting our beliefs by z_{t_0} alone will not help us to learn about the local behaviour of $u_{t_1}(x)$ around x^*. The information that z_{t_0} carries regarding x^*,may be exploited by adjusting beliefs about x^* directly by the system observation and then running the model at $\hat{x}$, thus learning about the local behaviour of $u_{t_1}(x)$ around x^*. The hat run method, therefore, allows us to use both global and local information regarding our computer model as part of a tractible forecast methodology.

Before computing a Bayes Linear calibrated forecast, there are a number of computational issues to overcome. In order to compute $\hat{x}$, the quantities $Var\left[z_{t_0}\right]$ and $E\left[z_{t_0}\right]$ must be obtained by integrating out x^* and computing the usual moments of y via (2.20) and (2.21). The other quantity required in order to compute $\hat{x}$, $Cov\left[x^*, z_{t_0}\right]$, would be obtained in the following way.

$$Cov\left[x^*, z_{t_0}\right]_{ki} = Cov\left[x_k^*, f_i(x^*)\right] = Cov\left[x_k^*, \beta_{ij} g_j(x^*) + u_i(x^*)\right],$$

if we assume that our emulator for $f(x)$ has the usual form (2.4). This means that

$$Cov\left[x^*, z_{t_0}\right]_{ki} = E\left[\beta_{ij}\right] Cov\left[x_k^*, g_j(x^*)\right] + Cov\left[x^*, u_i(x^*)\right]$$

and so in order to calculate this quantity, we require further numerical integration. The first half of this calculation may be readily calculated from the distribution $p(x^*)$, although the numerical integrations involved may be difficult. However, the

object $Cov\left[x_k^*, u_i(x^*)\right]$ is more complicated. By definition

$$\begin{aligned} Cov\left[x^*, u(x^*)\right] &= E\left[x^* u(x^*)\right] - E\left[x^*\right] E\left[u(x^*)\right] \\ &= E\left[E\left[x^* u(x^*)|x^*\right]\right] - E\left[x^*\right] E\left[E\left[u(x^*)|x^*\right]\right] \\ &= E\left[x^* E\left[u(x^*)|x^*\right]\right] - E\left[x^*\right] E\left[E\left[u(x^*)|x^*\right]\right] \end{aligned}$$

and in order to proceed further we must integrate $x^* E\left[u(x^*)|x^*\right]$ numerically, where $E\left[u(x^*)|x^*\right]$ will usually be a random field conditioned on the runs we have on f. This is feasible, in princple, and may be computed using the same order of magnitude computing power as is required in order to compute $Var\left[y\right]$.

The Bayes Linear calibrated forecast, (6.2) and (6.3), requires that we calculate the joint moments of y, z_{t_0} and $\hat{f}$. Consider, first, the computation of $E\left[\hat{f}\right]$. Suppose we write

$$\hat{x} = v + W z_{t_0}$$

where v and W are readily derived from (6.1). Then

$$\hat{f} = f(v + W z_{t_0}) = f(v + W(f_{t_0}(x^*) + \eta_{t_0} + e_{t_0})),$$

which, assuming the usual form for our emulator for f, may be written as

$$\hat{f} = \beta g(v + W(\beta_{t_0 j} g_j(x^*) + u_{t_0}(x^*) + \eta_{t_0} + e_{t_0})) + u(\hat{x}). \tag{6.4}$$

Now,

$$E\left[\hat{f}\right]_i = E\left[\beta\right]_{ij} E\left[E\left[g_j(v + W(\beta_{t_0 j} g_j(x^*) + u_{t_0}(x^*) + \eta_{t_0} + e_{t_0}))|x^*\right]\right] + E\left[E\left[u(\hat{x})|x^*\right]\right].$$

In order to evaluate this expression analytically, we would require specification of the moments of a number of complicated quantities as functions of the elements of β, η_{t_0}, e_{t_0} and $u_{t_0}(x)$, along with specification of the quantity $E\left[u(\hat{x})|x^*\right]$. In addition, and in order to know what the quantities that we must specify are, we must be able to expand the formula $g(v + W(\beta g(x^*) + u_{t_0}(x^*) + \eta_{t_0} + e_{t_0}))$ into a function of x^*, β, η_{t_0}, e_{t_0}, and $u_{t_0}(x)$. If we could do that, take expectations of the derived expression, specify the values of any moments required that we do not already have, and then integrate out x^*, we can compute $E\left[\hat{f}\right]$. The analytic expressions for $Var\left[\hat{f}\right]$ and $Cov\left[(\hat{f}, z_{t_0}), y\right]$ are even more complicated functions of the random

quantities x^*, β, η_{t_0}, e_{t_0}, $u_{t_0}(x)$, and $u(\hat{x})$. Unless the form of each of the functions in $g(x)$ was very simple, this would probably require the use of a computer algebra package. Goldstein and Rougier argue in [39] that the time involved in deriving the required moments using such a package and then making the required higher order specifications must be taken in the context of developing and running the computer simulator.

For important problems where our uncertainty about the model behaviour near x^* is significant, the information provided by a hat run may be too substantial to ignore, and the computational effort involved in producing an analytic calibrated forecast may be deemed worthwhile.

6.1.1 The hat run moments via sampling

A more attractive way to produce Bayes Linear calibrated forecasts is to compute the moments of $\hat{f}$ and the covariance between x^* and z_{t_0} directly using sampling methods. As an alternative to using an algebra package to find out the form of each of the quantities for which we require expectations and then specifying each of these individually, we could make distributional assumptions regarding η, e, x^*, β and $u(x^*)$, and then sample the required distributions directly. We now describe this sampling procedure in detail.

In the following discussion, we suppose that we have already observed runs $F = f(x_1), \ldots, f(x_n)$ so that our beliefs about the quantities in our emulator for the computer model have been adjusted by these runs. We know that

$$\hat{x} = v + W z_{t_0},$$

where v and W are functions of the moments of z_{t_0} and the matrix $Cov\,[x^*, z_{t_0}]$. Suppose we use our second order beliefs to characterize probability distributions for η, e, $\{\beta, u(x)\}|F$. Added to $p(x^*)$ this gives us distributions on each of the random quantities required to sample from the distribution of $\{\hat{f}, y, z_{t_0}\}$.

Our first goal will be to obtain v and W. We do this by sampling an x^*, a β, and a $u(x^*)$ in order to obtain a sample from $f(x^*)$. We then sample a value of η and a value of e_{t_0} in order to obtain a sampled value of z_{t_0}. Repeating this process

a large number of times we obtain a sample from the joint distribution of x^* and z_{t_0}. We may estimate $E[z_{t_0}]$, $Var[z_{t_0}]$ and $Cov[x^*, z_{t_0}]$ directly from this sample and use these to fix v and W.

In theory, we could now condition each of our distributions on these samples and then begin a new sampling experiment designed to sample from the joint distribution of $\hat{f}$, y and z_{t_0}. However, this may prove to be computationally demanding and we may prefer to assume v and W are now known and to ignore all previous samples and begin a new experiment.

We have that

$$\begin{aligned}\hat{f} &= \beta g(\hat{x}) + u(\hat{x}) \\ &= \beta g(v + W(\beta_{t_0 j} g_j(x^*) + u_{t_0}(x^*) + \eta_{t_0} + e_{t_0})) + u(\hat{x}),\end{aligned}$$

and we require a sample from $\{\hat{f}, y, z_{t_0}\}$. We may begin by sampling a value of x^* and values of β and $u(x^*)$. This allows us to compute a sample value of $f(x^*)$, which we shall denote $\tilde{f}(\tilde{x}^*)$. We may now sample values of η and e_{t_0} to obtain sample values of y and z_{t_0}. As we have already sampled a value of β, the only random quantity we have not sampled from in (6.4) is $u(\hat{x})$. Having observed a value of β and of $u(x^*)$, we have new information about $u(\hat{x})$. Effectively, $u(\hat{x})$ is now the random field conditioned on $\{F, \tilde{f}(\tilde{x}^*)\}$. In order to sample an appropriate value of $u(\hat{x})$, then, we must re-condition the random field $u(x)$ on $\{F, \tilde{f}(\tilde{x}^*)\}$.

As we have used second order beliefs about $u(x)$ to characterize a probability distribution, it is perhaps appropriate to perform a Bayes Linear update of these beliefs and then to use the new moments to characterize our probability distribution. Rather than adding $\tilde{f}(\tilde{x}^*)$ to the collection comprising F and adjusting the prior moments of $u(x)$ by this new, enlarged, collection in order to recondition the random field, it would be computationally efficient to perform a partial Bayes Linear adjustment.

We can use the partial Bayes Linear update equations (5.15), (5.16) and (5.17) to reduce the computational burden of the updating of the random field $u(\hat{x})|F$ by the extra point $\tilde{f}(\tilde{x}^*)$. That is, we can compute

$$E_{\tilde{f}(\tilde{x}^*)\cup F}[u(x)] = E_{[\tilde{f}(\tilde{x}^*)|F]}[u(x)] + E_F[u(x)]$$

and

$$Var_{\tilde{f}(\tilde{x}^*)\cup F}\left[u(x)\right] = Var_F\left[u(x)\right] - Var\left[E_{[\tilde{f}(\tilde{x}^*)|F]}\left[u(x)\right]\right],$$

where

$$E_{[\tilde{f}(\tilde{x}^*)|F]}\left[u(x)\right] = E_{\mathbb{A}_F(\tilde{f}(\tilde{x}^*))}\left[u(x) - E\left[u(x)\right]\right]$$

and

$$\mathbb{A}_F(\tilde{f}(\tilde{x}^*)) = \tilde{f}(\tilde{x}^*) - E_F\left[\tilde{f}(\tilde{x}^*)\right].$$

All of the adjustments to be made by the potentially large collection of data F are performed before the sampling experiment begins. This is so that adjustment by the extra run $\tilde{f}(\tilde{x}^*)$ does not require additional large matrix inversions. This will be important as, when obtaining a large sample, this calculation will turn out to be the most time consuming.

Having conducted a partial updating of the random field $u(\hat{x})$ by $\tilde{f}(\tilde{x}^*)$ given F, we may then re-characterize our distribution for $u(\hat{x})$ so that it has the correct covariance structure, draw a sample from the distribution, and compute $\hat{f}$. In summary, in order to produce one sample from the joint distribution of $\{\hat{f}, y, z_{t_0}\}$, given that we have already calculated and fixed v and W, we must do the following:

1. Draw samples from $p(x^*)$, $p(\eta)$, $p(e_{t_0})$, $p(\beta, u(x^*)|x^*, F)$ to obtain $\tilde{f}(\tilde{x}^*)$ and sample values of $\hat{x}$, y and z_{t_0};

2. Perform a partial Bayes Linear update of the random field $u(\hat{x})|F$ by $\tilde{f}(\tilde{x}^*)$ in order to characterize a new distribution for $u(\hat{x})$;

3. Draw a sample value for $u(\hat{x})$ in order to compute $\hat{f}$ using the already sampled values of β and $\hat{x}$.

Note that when generating multiple samples from $\{\hat{f}, y, z_{t_0}\}$, we do not remember the values of $\tilde{f}(\tilde{x}^*)$ from previous samples when performing the partial Bayes Linear update step. At the beginning of each sample, we presume that $u(x)$ is a random field conditioned on F only and that β is correlated with this random field.

We note that many of the decisions we might make in specifying expectations of functions of the random quantities $\{x^*, \eta, e, \beta, u(x^*), u(\hat{x})\}$, having used a computer algebra package to derive the form of the required functions, would probably be

based on distributional assumptions. Goldstein and Rougier show in [39] that if, for example, $g(x) = x$ then in order to compute an expression for $Var\left[\hat{f}\right]$, we must specify third and fourth order moments of β. A natural way of specifying these is to imagine a distributional form for β (in the paper they choose a normal distribution) and derive higher order moments from that. In this case, the large sampling experiment will not only be faster to perform than the more laborious use of a computer algebra package and individual specification of a large number of expectations, but will be at least as accurate. Furthermore, the sampling scheme avoids the sticky problem of specifying quantities such as $E\left[u(\hat{x})|x^*\right]$, which is a very difficult quantity to think about. We shall discuss this last issue in detail in section 6.2.5.

Our argument, then, is that in order to perform a Bayes Linear calibrated forecast via a hat-run, calculating the required moments for the adjustment is most efficiently done via a large sampling experiment such as the one we have described. We now turn to the case that we are interested in, namely where the computer model is a function of both model inputs x and decisions θ, and look at how we may arrive at decision-dependent calibrated forecasts and how they may be used to provide policy support within our Sequential Emulation framework.

6.2 The Hat function

We return to the case that we have described for the majority of this thesis, where the computer model f is a function of model inputs x and decision variables θ. If our system data z_{t_0} contains significant information regarding x^* then, for any possible θ, our decision-dependent forecasts may not be as accurate as they could be. This will be the case if we have captured the global effects of the model well, but where we still have a lot of uncertainty regarding local behaviour of the model. If the information in z_{t_0} regarding x^* is substantial, then that information could point us to an area of the domain of the model that we could explore in order to resolve relevant local uncertainty.

In order to ensure that this information is included, we desire to make calibrated

forecasts. However, as we have discussed in previous chapters, when the ultimate goal is to provide decision support for policy makers, these forecasts must be relatively quick as we will be required to compute a large number of them. The current, fully probabilistic, methods of calibrated prediction are, as we discussed in chapter 2, too computationally expensive to perform a large number of times in big problems. Whilst we may use the fully probabilistic methods as part of a multi-level emulation of the decision tree, we also require tractible methods to make this technology work. This was discussed in section 3.3.4. We are motivated then, to explore ways that the Bayes Linear calibrated prediction methodology can be applied to models with decision inputs.

The hat run methodology allows us, in principle, to learn about local behaviour of the model around x^*. This learning is achieved by running the model at our 'best guess' for the location of x^*, namely $\hat{x}$. Even for very expensive computer models with long run times, the requirement of one further evaluation of the model in order to significantly improve our forecasts does not seem unreasonable. If we were to extend the hat run methodology to obtain decision-dependent calibrated forecasts, we would require a different model run for each forecast. This limits the speed of our forecasting methodology by the speed of running the computer model. This means that, for models with very long run times, Bayes Linear calibrated decision-dependent forecasting would be infeasible because we would not be able to generate enough forecasts in order to capture our beliefs across the decision space.

6.2.1 The Hat spin

An exception to the above case occurs within the class of models that require spin up. We described the process of spin up as it applied to C-GOLDSTEIN in section 4.2. Most climate simulators that we have encountered, including the large ones at the Hadley centre HadCM3 and HadGEM, require spinning up for a model time consisting of thousands of years before a particular policy may be explored. In the case where, for a particular choice of model inputs x, the model must be spun up before the decisions are added, and when the spin up time accounts for the majority of the model run-time, the extension of the hat run methodology to computer

simulators with decision variables may be a useful tool.

The idea is to locate $\hat{x}$ in the usual way and then to perform a *hat spin* by spinning the model up at $\hat{x}$. We then define the *hat funtion*, $\hat{f}(\theta)$, to be

$$\hat{f}(\theta) = f(\hat{x}, \theta). \tag{6.5}$$

When spin up is time consuming relative to running the model into the future for a particular value of θ, we may obtain many evaluations of the hat function relatively cheaply. Although the time taken to compute a decision-dependent forecast will still be at least as long as it takes to run the model from its spun up state into the future, this may be fast enough to allow us to provide useful decision support.

If the computer model takes a very long time to run, then, even if we perform the hat spin, the hat function may take a long time to evaluate. For example, suppose that we have a computer model that takes one month to run for one choice of its inputs. Further, suppose that the ratio of time taken to spin up, compared with the time taken to run the model into the future for given θ after spin up is the same as it is for C-GOLDSTEIN, about $40 : 1$. Then, having performed the hat spin, each evaluation of the hat function would still take approximately 18 hours (based on a 30 day month). In these cases, we may try simultaneous computing to obtain enough evaluations to emulate the hat function. If we could build an accurate emulator for the hat function, the time taken to compute a decision-dependent forecast could be very fast indeed.

6.2.2 Forecasting via the hat function

Future states of the complex system y will depend on our choice of θ, hence our model is $f(x, \theta)$. However, the historical system observations to which we have access have no dependence on θ. Hence,

$$\hat{x} = v + W z_{t_0}$$

does not depend on θ. Therefore, the hat spin, which defines a hat function $\hat{f}(\theta)$, does not depend on θ and is only required once. This means that the calculations required to obtain v and W are needed only once. Having obtained $\hat{x}$ and performed

the hat spin up, however, many of the quantities required for the forecast must be computed for each individual θ, including the hat function itself.

A Bayes Linear calibrated decision-dependent forecast of $y(\theta)$ given data z_{t_0} and the hat function $\hat{f}(\theta)$ is the pair

$$E_{z_{t_0},\hat{f}(\theta)}[y] = E[y] + Cov\left[y,(z_{t_0},\hat{f}(\theta))\right] Var\left[z_{t_0},\hat{f}(\theta)\right]^{-1}\left((z_{t_0},\hat{f}(\theta)) - E\left[z_{t_0},\hat{f}(\theta)\right]\right), \tag{6.6}$$

and

$$Var_{z_{t_0},\hat{f}(\theta)}[y] = Var[y] - Cov\left[y,(z_{t_0},\hat{f}(\theta))\right] Var\left[z_{t_0},\hat{f}(\theta)\right]^{-1} Cov\left[(z_{t_0},\hat{f}(\theta)),y\right]. \tag{6.7}$$

In order to compute the forecast, we require an evaluation of the hat function as well as second order moments of the collection $\{y, z_{t_0}, \hat{f}(\theta)\}$ for any θ. We have

$$\begin{aligned}\hat{f}(\theta) &= \beta g(v + W(f_{t_0}(x^*) + \eta_{t_0} + e_{t_0}), \theta) + u(\hat{x},\theta)\\ &= \beta g(v + W(\beta_{t_0 j} g_j(x^*,\theta) + u_{t_0}(x^*) + \eta_{t_0} + e_{t_0}), \theta) + u(\hat{x},\theta).\end{aligned} \tag{6.8}$$

However, note that the term $\beta_{t_0 j} g_j(x^*,\theta)$ does not depend on θ because $f_{t_0}(x^*,\theta) \equiv f_{t_0}(x^*)$. This is because historical values of the system do not depend on policies we make today in order to affect future states. We allow $g(x,\theta)$ to contain all of the regression terms required to describe any output of the simulator in order to provide structure for our calculations. However, the matrix β is such that the submatrix containing the rows of β corresponding to time t_0 has column i fixed at 0 if $g_i(x,\theta)$ is an expression involving θ. Therefore, if $g(x,\theta)$ is separable in x and θ, then we may write

$$\hat{f}_i(\theta) = \beta_{ip}\left(h_p(v + W(\beta_{t_0 j} g_j(x^*,0) + u_{t_0}(x^*) + \eta_{t_0} + e_{t_0})) * k_p(\theta)\right) + u_i(\hat{x},\theta).$$

We now present the integrals that must be evaluated numerically in order to obtain the moments of $\hat{f}(\theta)$ and the covariance between $\hat{f}(\theta)$ and $\{y, z_{t_0}\}$. We write these integrals down as expectations, variances and covariances of quantities conditioned on x^*. From this point on then, any expectation, variance, or covariance involving only quantities that are specifically conditioned on x^* imply moments taken with respect to the distribution $p(x^*)$ via numerical integration or otherwise. For example,

we regard the quantity $E\left[u(\hat{x},\theta)|x^*\right]$ as a quantity specifically conditioned on x^* so that

$$E\left[E\left[u(\hat{x},\theta)|x^*\right]\right] \equiv \int E\left[u(\hat{x},\theta)|x^*\right] p(x^*)dx^*.$$

We use this convention to simplify the expressions that arise from these calculations, and because it will be useful to think of the required moments as complicated functions of x^* that must first be evaluated before numerically integrating with respect to $p(x^*)$.

We know that

$$\begin{aligned} E\left[\hat{f}(\theta)\right] =& E\left[E\left[\beta g(v + W(\beta_{t_0 j} g_j(x^*,\theta) + u_{t_0}(x^*) + \eta_{t_0} + e_{t_0}),\theta)|x^*\right]\right] \\ &+ E\left[E\left[u(\hat{x},\theta)|x^*\right]\right] \end{aligned} \tag{6.9}$$

and

$$Var\left[\hat{f}(\theta)\right] = Var\left[E\left[\hat{f}(\theta)|x^*\right]\right] + E\left[Var\left[\hat{f}(\theta)|x^*\right]\right]. \tag{6.10}$$

We also have

$$Cov\left[\hat{f}(\theta), z_{t_0}\right] = Cov\left[\hat{f}(\theta), y_{t_0}\right] + Cov\left[\hat{f}(\theta), e_{t_0}\right],$$

so that we can obtain $Cov\left[\hat{f}(\theta), z_{t_0}\right]$ directly from $Cov\left[\hat{f}(\theta), y\right]$ if we can compute $Cov\left[\hat{f}(\theta), e_{t_0}\right]$.

$$\begin{aligned} Cov\left[\hat{f}(\theta), y\right] &= Cov\left[\hat{f}(\theta), f(x^*,\theta) + \eta\right] \\ &= Cov\left[\hat{f}(\theta), f(x^*,\theta)\right] + Cov\left[\hat{f}(\theta), \eta\right] \\ &= E\left[\hat{f}(\theta) f(x^*,\theta)\right] - E\left[\hat{f}(\theta)\right] E\left[f(x^*,\theta)\right] + E\left[\hat{f}(\theta)\eta\right] - E\left[\hat{f}(\theta)\right] E\left[\eta\right] \\ &= E\left[E\left[\hat{f}(\theta) f(x^*,\theta)|x^*\right]\right] - E\left[E\left[\hat{f}(\theta)|x^*\right]\right] E\left[E\left[f(x^*,\theta)|x^*\right]\right] \\ &\quad + E\left[E\left[\hat{f}(\theta)\eta|x^*\right]\right] - E\left[E\left[\hat{f}(\theta)|x^*\right]\right] E\left[\eta\right] \\ &= E\left[Cov\left[\hat{f}(\theta), f(x^*,\theta)|x^*\right]\right] + E\left[E\left[\hat{f}(\theta)|x^*\right] E\left[f(x^*,\theta)|x^*\right]\right] \\ &\quad - E\left[E\left[\hat{f}(\theta)|x^*\right]\right] E\left[E\left[f(x^*,\theta)|x^*\right]\right] + E\left[Cov\left[\hat{f}(\theta), \eta|x^*\right]\right] \\ &\quad + E\left[E\left[\hat{f}(\theta)|x^*\right] E\left[\eta|x^*\right]\right] - E\left[\eta\right] E\left[E\left[\hat{f}(\theta)|x^*\right]\right] \\ &= E\left[Cov\left[\hat{f}(\theta), f(x^*,\theta)|x^*\right]\right] + Cov\left[E\left[\hat{f}(\theta)|x^*\right], E\left[f(x^*,\theta)|x^*\right]\right] \\ &\quad + E\left[Cov\left[\hat{f}(\theta), \eta|x^*\right]\right]. \end{aligned} \tag{6.11}$$

The two ways of evaluating means, variances and covariances on the collection $\{\hat{f}(\theta), y, z_{t_0}\}$ are the same as for the hat run. We may either use a computer algebra package to expand each of the conditional means, variances and covariances above into a linear combination of expectations that are functions of θ, β, x^*, η, e, $u(x^*, \theta)$, and $u(\hat{x}, \theta)$, specify the expectations of any quantities for which we have not yet stated beliefs and then integrate with respect to x^*; or we can specify distributions on x^*, η, e, $u(x^*, \theta)$, and β and perform a large scale sampling experiment as we would for a normal hat run.

6.2.3 Sampling the hat function moments

To obtain the moments of the hat function via a large scale sampling experiment for given θ, we proceed in precisely the same fashion as we would for a single hat run. Having fixed the matrices v and W (which do not depend on θ) via an initial sampling experiment, we sample values of x^*, β, $u(x^*, \theta)$, η, and e to obtain values of $\hat{x}$, $y(\theta)$ and z_{t_0}. We then perform a partial Bayes Linear adjustment of $u(x, \theta)$ by the sampled $f(x^*, \theta)$ given F, in order to characterize a new distribution for the random field $u(\hat{x}, \theta)$ from which we may then sample to obtain a $\hat{f}(\theta)$.

This sampling experiment must be repeated a very large number of times in order to accurately estimate the required quantities. The time taken to perform each sample will not be negligible because, even if the distributions we have specified for each of our quantities is easily sampled from, we must recondition the random field $u(x, \theta)$ at each step.

If we had only a single hat run for which to obtain the required moments, the time taken to perform the sampling experiment would probably not concern us. Taken within the context of running the simulator at $\hat{x}$ and if the problem were serious enough to require calibrated forecasting, the time taken to obtain the moments of $\hat{f}$ by sampling would be worthwhile. This argument, however, is very unlikely to hold when forecasting via a hat function.

We stated earlier that the hat function will be particularly useful for obtaining calibrated forecasts with simulators that require spin up. For such models, the most expensive part of the calculation, as far as the simulator is involved, is spinning

the model up. Having spun the model up, evaluations of the hat function $\hat{f}(\theta)$ are relatively cheap and, as they contain valuable information regarding the local behaviour of the simulator around x^* for any θ, our methods of forecasting should be able to compete with the speed of an evaluation of the hat function. If this is not the case, we would potentially be wasting evaluations of the hat function that we can make, or that have been made, because our methods are not fast enough to deal with them.

Unless, having spun the model up, the simulator still takes a considerable amount of time to complete a run, directly sampling from the distribution of $\{\hat{f}(\theta), y(\theta), z_{t_0}\}$ in order to obtain the required forecasting moments is likely to take too long. If it does take a very long time to evaluate the hat function, we may have to emulate it anyway in order to perform a Sequential Emulation. We discuss this further in section 6.2.6.

If running the model into the future after spin up is significantly quicker than performing the sampling experiment to calculate the hat function moments, then we must either emulate $E\left[\hat{f}(\theta)\right]$, $Var\left[\hat{f}(\theta)\right]$, $Cov\left[\hat{f}(\theta), y\right]$ and $Cov\left[\hat{f}(\theta), z_{t_0}\right]$, or else abandon the sampling method altogether. Our ultimate goal is to add calibrated forecasting via the hat function to our Sequential Emulation methodology. The idea of using an emulator to calculate the required moments fits into this framework naturally, as a particular type of fast forecast used in coarse calculations of our expected loss. The conceptual simplicity of this method also increases its appeal.

Emulating the hat function moments as a function of θ is a daunting task. Suppose our computer model has three outputs and that one of these corresponds to historical system values. Then the number of individual scalar emulators required is

$$3 + 3^2 + 3^2 + 3 = 24.$$

Suppose, for argument's sake, that we treat this as one large 24-output function to be emulated at once. Not only would we require a vast number of separate sampling experiments in order to build such an emulator, but we would inevitably introduce a lot of new uncertainty into our forecasts. This new uncertainty would be in the form of uncertainty regarding the new emulator coefficients and processes. The goal

of this type of calibrated forecasting is to resolve local uncertainty around x^* for any θ. We should, therefore, be wary of any method that adds to our uncertainty, particularly if this can be avoided.

6.2.4 Using a computer algebra package

By using a computer algebra package, much of the calculation required to compute the moments of the hat function can be done exactly. In contrast to the direct sampling method, where a large collection of $\{\hat{f}(\theta), y, z_{t_0}\}$ are generated and the required moments are computed directly for the sample, calculations with a computer algebra package involve computing all of the quantities that are conditioned on x^* in the expressions for $E\left[\hat{f}(\theta)\right]$, $Var\left[\hat{f}(\theta)\right]$ and $Cov\left[\hat{f}(\theta), y\right]$, given by (6.9), (6.10), and (6.11), exactly. We use the algebra package to evaluate the required expressions and then integrate x^* out numerically as with all of our forecasting methods so far. Therefore, if we are able to use a computer algebra package to derive easy to evaluate expressions in x^*, $u(x^*, \theta)$ and θ, we turn the calibrated forecasting problem into the familiar forecasting problem, where the main task is integrating out x^* from given expressions involving those quantities for any θ. We may then use the methods discussed in section 2.3.2 to obtain forecasts at whatever level of accuracy we deem suitable.

The conditional moments that the algebra package must handle are $E\left[\hat{f}(\theta)|x^*\right]$, $Var\left[\hat{f}(\theta)|x^*\right]$, $Cov\left[\hat{f}(\theta), f(x^*, \theta)|x^*\right]$, $Cov\left[\hat{f}(\theta), \eta|x^*\right]$, and $Cov\left[\hat{f}(\theta), e_{t_0}|x^*\right]$.

$$\begin{aligned} E\left[\hat{f}(\theta)|x^*\right] =& E\left[\beta g(v + W(\beta_{t_0 j} g_j(x^*, 0) + u_{t_0}(x^*) + \eta_{t_0} + e_{t_0}), \theta)|x^*\right] \\ &+ E\left[u(\hat{x}, \theta)|x^*\right]. \end{aligned}$$

We require a computer algebra package so that we may evaluate the vector of expressions

$$v + W(\beta_{t_0 j} g_j(x^*, 0) + u_{t_0}(x^*) + \eta_{t_0} + e_{t_0}),$$

which is a vector of the same length as x, and then apply $g(\cdot, \cdot)$ to this vector in order to compute the vector of expressions

$$\beta g(v + W(\beta_{t_0 j} g_j(x^*, 0) + u_{t_0}(x^*) + \eta_{t_0} + e_{t_0}), \theta).$$

This is a vector of expressions, each element of which is a linear combination of known functions of the random quantities β_{ij} (for all ij appropriate to the dimension of β), $u_{t_0}(x^*)$, η_{t_0}, e_{t_0} and x^*, and the known quantities v, W and θ. $E\left[\hat{f}(\theta)|x^*\right]$ is now a linear combination of expectations of each of these functions plus $E\left[u(\hat{x},\theta)|x^*\right]$.

By using the same distributional assumptions as we were prepared to use to conduct a large sampling experiment, or otherwise, we can either compute or write down each of these expectations as functions of x^* and $E\left[\zeta(u_{t_0}(x^*))|x^*\right]$, where $\zeta(\cdot)$ is any function of the random field $u_{t_0}(x^*)$, for example, $\beta_{12}u_{t_0}(x^*)$. The algebra package would be used to compute the expressions, and then the user must provide a set of rules for evaluating and simplifying the expressions. For example, we might decide that $E\left[a(\eta_{t_0})b(\beta)|x^*\right] \equiv 0$ for any non-constant functions $a(\cdot)$ and $b(\cdot)$. Note that we still have not specified how to deal with $E\left[u(\hat{x},\theta)|x^*\right]$.

The use of the computer algebra package to obtain the other moments of $\hat{f}(\theta)|x^*$ is similar. Each variance or covariance would first be written in terms of expectations. For example,

$$Var\left[\hat{f}(\theta)|x^*\right] = E\left[\hat{f}(\theta)^2|x^*\right] - E\left[\hat{f}(\theta)|x^*\right]^2.$$

Then the expression for $\hat{f}(\theta)^2$ is computed by expanding out

$$(\beta g(v + W(\beta_{t_0 j} g_j(x^*, 0) + u_{t_0}(x^*) + \eta_{t_0} + e_{t_0}), \theta) + u(\hat{x}, \theta))^2$$

and breaking it down into a matrix, each of whose elements is a sum of expectations that are then computed as functions of x^*, $E\left[\zeta(u_{t_0}(x^*))|x^*\right]$, θ and quantities involving $u(\hat{x},\theta)$ using a set of rules specified by the user.

Ignoring expressions involving $u(\hat{x},\theta)$ for the moment, this method would not involve as much work as it may first appear. The cost and difficulty in developing the code for an algebra package to expand these expressions into the required linear combination of expectations must only be undertaken once. Specifying a set of rules for evaluating most of these expectations to leave just functions of x^*, θ and the expectation of any function of $u_{t_0}(x^*)$ will only be required once. Furthermore, these rules may be derived from the distributional assumptions we would have made for the conceptually simpler sampling experiment method. Once we have achieved

these things, and assuming we can handle any functions involving $u(\hat{x}, \theta)$ in some way, we are left with an expression that can be evaluated for any x^*, θ and some assumptions regarding the functions of $u_{t_0}(x^*)$, relatively cheaply. We could, for example, choose a distribution for $u_{t_0}(x^*)|x^*$, so that each of the expectations of the functions of $u_{t_0}(x^*)$ given x^* could be derived from this distribution.

Having expended this initial, one-off, effort, we can compute the hat function moments for any θ by integrating x^* out of these expressions numerically. Although this may be challenging, it is no more so than providing a normal forecast by integrating out x^* once the algebra package has done its work. Therefore, if we can specify a way of handling moments of $u(\hat{x}, \theta)$, we can use a computer algebra package to obtain the required moments of $\hat{f}(\theta)$ more quickly and more accurately than via direct sampling, after an initial set up cost. This claim is justified because, having used the algebra package, we are essentially left with a sampling experiment that involves sampling from $p(x^*)$ as opposed to the larger scale experiment described in the previous section. We, therefore, reduce sampling error and require far fewer individual random number generations to compute a forecast.

6.2.5 Dealing with $u(\hat{x}, \theta)$

Expectations involving functions of $u(\hat{x}, \theta)$ conditioned on x^* pose particular challenges for the computer algebra package method of calibrated forecasting. The issue lies in the fact that $\hat{x}$ depends on the random field $u_{t_0}(x^*)|F$ and the value of β conditioned on this field. This means that the random field $u(\hat{x}, \theta)|x^*$ is implicitly conditioned on a, potentially, very complicated function of another random field. Therefore, for a given x^*, we cannot say a great deal about the moments of $u(\hat{x}, \theta)$ without further information.

In the paper by Goldstein and Rougier [39], where they explicitly attempt to write down the expressions for the hat run moments for the case $g(x) = x$, they handle functions of $u(\hat{x})$ by making the following assumption, which we shall generalize to functions of $u(\hat{x}, \theta)$. Let $\zeta(\cdot)$ be some function of $u(\hat{x}, \theta)$ required for computing the hat function moments, then

$$E\left[\zeta(u(\hat{x}, \theta))|x^*\right] \approx E\left[\zeta(u(\hat{x}^*, \theta))|x^*\right], \tag{6.12}$$

where

$$\hat{x}^* = E\left[\hat{x}|x^*\right].$$

For any x^*, $\hat{x}^*$ can be computed directly from (6.1) and, therefore, we can compute all of the expectations involving $u(\hat{x},\theta)$ by integrating out x^* in the usual way, as long as we believe this assumption.

The problem is that there is no reason to believe this assumption, or even to know whether it is reasonable or badly wrong. We know that $\hat{x}$ is a point in X, whose location depends on x^*. The nature of this dependence is complicated and uncertain, however we do have beliefs about it and we do have an expectation for its location, namely $\hat{x}^*$. However, $u(\hat{x},\theta)$ is a complicated random field, our beliefs about which may be very different at $\hat{x}^*$ from other plausible values of $\hat{x}$, depending on the locations of the runs comprising F in X and x^*.

Suppose, for example, that one of our runs is at, or very close to, $\hat{x}^*$ and suppose our covariance function for $u(x,\theta)$ has very short correlation lengths. If we have no other points close to $\hat{x}^*$ and the observation of the residual surface made at $\hat{x}^*$ was very large compared to the prior expectation for the residual, which here we assume to be 0 for all x, then we would have a sharp spike in the random field for $u(x,\theta)$. Now suppose that given x^*, we believed that $\hat{x}$ was equally likely to be anywhere in a large neighbourhood of $\hat{x}^*$. Then our true expectation of $u(\hat{x},\theta)|x^*$ would be close to zero and only slightly influenced by the large spike at $\hat{x}^*$, and $E\left[u(\hat{x}^*,\theta)|x^*\right]$ would be much larger than our true expectation.

In order to be more precise, we might consider the relation

$$E\left[\zeta(u(\hat{x},\theta))|x^*\right] = E\left[E\left[\zeta(u(\hat{x},\theta))|\hat{x},u_{t_0}(x^*),x^*\right]\right],$$

with the outer expectation taken with respect to $p(\hat{x},u_{t_0}(x^*)|x^*)$. Theoretically this might be solved using

$$E\left[\zeta(u(\hat{x},\theta))|x^*\right] = \int\int E\left[\zeta(u(\hat{x},\theta))|\hat{x},u_{t_0}(x^*),x^*\right]p(\hat{x}|u_{t_0}(x^*),x^*)d\hat{x}p(u_{t_0}(x^*)|x^*)du_{t_0}(x^*),$$

if suitable distributions $p(\hat{x}|u_{t_0}(x^*),x^*)$ and $p(u_{t_0}(x^*)|x^*)$ may be specified. Each $u_{t_0}(x^*)$ is informative for the random field $u(x,\theta)|F$ and for $\beta|F$. To solve this integral numerically for a given x^*, we would sample a value of $u_{t_0}(x^*)$ and partially

adjust beliefs about $u(x, \theta)$ and β by this quantity given F. If we had distributions on each of η_{t_0} and e_{t_0} in addition, we could then draw samples from $p(\hat{x}|u_{t_0}(x^*), x^*)$ and compute $E\left[\zeta(u(\hat{x}, \theta))|\hat{x}, u_{t_0}(x^*), x^*\right]$ for each sample. To compute the required moments of the hat function using these ideas would involve the same kind of large scale sampling experiment we have been trying to avoid through using the algebra package.

Our original idea for computing the hat function moments involved generating a very large sample of $\{\hat{f}(\theta), y, z_{t_0}\}$, computing the required moments from the sample, and repeating the process for enough values of θ to allow us to build an emulator for them. One of the principal issues with this was the amount of uncertainty being introduced to a calculation that was designed to resolve local uncertainty. By using a combination of this original idea along with the ideas involving exact computations with a computer algebra package, we can build an emulator for the hat function moments whilst introducing a minimal amount of uncertainty through emulation.

Suppose we complete the steps of section 6.2.4 to obtain all of the elements of the required expectations conditioned on x^* that do not involve $u(\hat{x}, \theta)$, as easy to evaluate expressions in θ and x^*. Suppose then, that for each of these integrals we exploit any separability in x^* and θ, and we solve analytically any parts of the calculation that we can. Let $M(\theta)$ be the collection of all of the hat function moments that we require. The steps taken so far leave us with

$$M(\theta) = a(\theta) + b(x^*, \theta) + c(u(\hat{x}, \theta)),$$

where $a(\theta)$ is a cheap function of θ derived from our calculations with the computer algebra package and any analytic integrations with respect to $p(x^*)$ we have been able to perform; $b(x^*, \theta)$ is a function that is computed by integrating cheap functions of x^*, $u_{t_0}(x^*)$ and θ numerically with respect to $p(x^*)$; and $c(u(\hat{x}, \theta))$ is a function of expectations of functions of $u(\hat{x}, \theta)$ derived from work done with the algebra package.

To evaluate $M(\theta)$ for given θ, computing $a(\theta)$ is trivial; computing $b(x^*, \theta)$ will involve either a large sample of x^* or another numerical integration technique; and computing $c(u(\hat{x}, \theta))$ will involve sampling a large number of $u(\hat{x}, \theta)$ by generating a sample from the collection $\{x^*, u_{t_0}(x^*), \beta, \eta_{t_0}, e_{t_0}\}$ each time, conditioning $u(x, \theta)$ on the sampled $u_{t_0}(x^*)$ and β, and then generating a $u(\hat{x}, \theta)$. If we assume that any

time we require a calibrated forecast, we are prepared to evaluate $b(x^*, \theta)$ directly (as we have in our other forecasting methodologies), then instead of emulating all of $M(\theta)$, we could emulate $c(u(\hat{x}, \theta))$ as a function of θ and, compared with the full sampling method described in section 6.2.3, provide a more accurate emulator for $M(\theta)$ that introduces much less uncertainty into our forecasting calculations.

6.2.6 The practicalities of using a computer algebra package

In order to use a computer algebra package to provide a more accurate emulator of the hat function moments, there is an initial set up cost and a number of problem dependent issues to overcome. However, the method is certainly feasible. We demonstrate this in Appendix F by providing example code for the computer algebra package Maple [70]. Alongside describing the steps that must be taken with an algebra package and describing what our Maple code does, we provide an illustration of our method using a simple example of an emulator for a computer model to derive the required expression for $E\left[\hat{f}(\theta)\right]$ only.

Suppose we have a computer model with two outputs, x and θ are univariate, and that only the second output depends on θ. Our emulator for this model is

$$f_1(x) = x + u_1(x)$$
$$f_2(x, \theta) = 3x + \theta + u_2(x, \theta),$$

so that $g(x, \theta) = (x, \theta)^T$ and we assume, for simplicity, that the coefficients are known. The first task for a computer algebra package would be to expand the expression for $\hat{f}(\theta)$ into a linear combination of the different quantities that we have. Our Maple code does this via the function *Fhat* described in appendix F on page 350. We do this by hand for our example here.

$$\hat{f}_1(\theta) = v + Wx^* + Wu_1(x^*) + W\eta_1 + We_1 + u_1(\hat{x})$$
$$\hat{f}_2(\theta) = 3v + 3Wx^* + 3Wu_1(x^*) + 3W\eta_1 + 3We_1 + \theta + u_2(\hat{x}, \theta).$$

The next step for the computer algebra package is to compute all required functions of these two expressions and expand them into a linear combination of quantities. For example, if we required $Var\left[\hat{f}(\theta)\right]$, we would require the algebra package to evaluate and expand each element of the matrix $\hat{f}(\theta)\hat{f}(\theta)^T$. In our Maple code this

step is automated by defining rules for expressing variances and covariances in terms of expectations. We describe this automation in appendix F on page 348

Next the algebra package must express each of the required hat function moments conditioned on x^* as the sum of expectations of unknown quantities conditioned on x^*. Our Maple code generates expressions for $E\left[\hat{f}(\theta)|x^*\right]$, $Var\left[\hat{f}(\theta)|x^*\right]$, $Cov\left[\hat{f}(\theta), y|x^*\right]$ and $Cov\left[\hat{f}(\theta), e_{t_0}|x^*\right]$ in terms of expectations of functions of x^*, θ, $u_{t_0}(x^*)$, β, η, e_{t_0} and $u(\hat{x}, \theta)$, for any form of $g(\cdot, \cdot)$. This is done via the functions *ExpectFhat*, *VarFhat*, *CovFhatY* and *CovFhate* in appendix F on page 351. In our example this step leaves us with

$$E\left[\hat{f}_1(\theta)|x^*\right] = v + Wx^* + WE\left[u_1(x^*)|x^*\right] + WE\left[\eta_1\right] + WE\left[e_1\right] + E\left[u_1(\hat{x})|x^*\right],$$

and

$$E\left[\hat{f}_2(\theta)|x^*\right] = 3v+3Wx^*+3WE\left[u_1(x^*)|x^*\right]+3WE\left[\eta_1\right]+3WE\left[e_1\right]+\theta+E\left[u_2(\hat{x},\theta)|x^*\right].$$

Having completed this initial step, the next job would be to assign a number of rules in order to evaluate many of these expectations, leaving only quantities that depend on x^*, $u_{t_0}(x^*)$, θ and $u(\hat{x}, \theta)$. In appendix F we give some example code for this. We present a piece of code that sets all expectations containing at least one element of each of the two distinct sets $\{\beta, u_{t_0}(x^*), u(\hat{x}, \theta)\}$ and $\{\eta, e_{t_0}\}$ to be zero, and expectations of functions of both η and e_{t_0} to be zero as well. This is on page 353. In addition to this, we demonstrate a piece of code that fixes all moments involving β alone, if $g(\cdot, \cdot)$ contains only monomials in x, by assuming that β has a normal distribution and using a characteristic function argument. This is demonstrated on page 354. To illustrate this step in our example, suppose that our discrepancy and observational error had zero mean, then we would have

$$E\left[\hat{f}_1(\theta)|x^*\right] = v + Wx^* + WE\left[u_1(x^*)|x^*\right] + E\left[u_1(\hat{x})|x^*\right],$$

and

$$E\left[\hat{f}_2(\theta)|x^*\right] = 3v + 3Wx^* + 3WE\left[u_1(x^*)|x^*\right] + \theta + E\left[u_2(\hat{x},\theta)|x^*\right].$$

Having used the algebra package and provided rules for evaluating as many expectations as possible, the output would then be written out to a file. This file would then

be read by the program with which we intend to perform numerical integration with respect to x^* and in whose environment we intend to compute calibrated forecasts by emulating those features of the hat function moments still depending on $u(\hat{x}, \theta)$.

The next task would be to integrate out x^* in order to remove the conditioning and to obtain the hat function moments. Using the notation of the previous section, we have

$$E\left[\hat{f}_i(\theta)\right] = a_i(\theta) + b_i(x^*, \theta) + c_i(u(\hat{x}, \theta))$$

where $a(\theta)$ is a cheap function of θ derived from our calculations with the computer algebra package and any analytic integrations with respect to $p(x^*)$ we have been able to perform; $b(x^*, \theta)$ is a function that is computed by integrating cheap functions of x^*, $u_{t_0}(x^*)$ and θ numerically with respect to $p(x^*)$; and $c(u(\hat{x}, \theta))$ is a function of expectations of functions of $u(\hat{x}, \theta)$ derived from work done with the algebra package. Assuming that we have $E[x^*]$ as it was required in order to compute $\hat{x}$, then in our example

$$a(\theta) = \begin{pmatrix} v + WE[x^*] \\ 3v + 3WE[x^*] + \theta \end{pmatrix},$$

$$b(x^*, \theta) = \begin{pmatrix} WE[E[u_1(x^*)|x^*]] \\ 3WE[E[u_1(x^*)|x^*]] \end{pmatrix},$$

and

$$c(\hat{x}, \theta) = \begin{pmatrix} E[E[u_1(\hat{x})|x^*]] \\ E[E[u_2(\hat{x}, \theta)|x^*]] \end{pmatrix}.$$

We would use the normal numerical integration methods of chapter 2 to compute $b(x^*, \theta)$, and would use the sampling experiment described in section 6.2.5 in order to emulate $c(\hat{x}, \theta)$.

Although there may be a deal of extra work involved in using a computer algebra package to help emulate the hat function moments in this way, this work may be undertaken during the time it takes to spin up the computer model. Having performed the emulation, we can then exploit spin up by evaluating the hat function anywhere and being able to to compute a Bayes Linear calibrated forecast in roughly the same amount of time it would take to provide an ordinary forecast. Note that this is limited by the time it takes us to integrate out x^*.

Certain, very powerful, computer models may have run times that are dominated by spin up time, but with such a long run time that the hat function takes a very long time to evaluate. In this case, the hat function itself may have to be emulated. A more interesting solution may involve an emulator for the forecast for $y(\theta)$, or some function of a forecast such as the expectation integrals required in Sequential Emulation, being built using ordinary forecasts as coarse evaluations, and a handful of calibrated forecasts as careful evaluations. Even in this case, the substantial amount of time required to spin the model up should provide enough time to use an algebra package in order to maximise what is learned from the few hat function evaluations for which we have time.

6.3 Sequential Emulation with the hat function

We consider the policy problem of section 3.1.3, where we must make policy today and have the option of making m interventions to that strategy at m distinct time points, each following an observation of the system. There are, essentially, two ways of introducing calibrated forecasting by amending the Sequential Emulation algorithm. We may choose to define $\hat{x} = v + Wz_{t_0}$, perform the hat spin, and allow our forecasts for y and for $z_{t_1}, \ldots, z_{t_m}$ to be calibrated via the hat function, $\hat{f}(\theta)$, which may be evaluated now for any θ. At each step in the Sequential Emulation algorithm, our forecasts only differ from those used in the original version of section 3.6 in that local information around x^* provided by z_{t_0} is accounted for.

The rather more abstract alternative to this method is to consider that each observation, z_{t_k}, will define its own $\hat{x}_{t_k} = v^k + W^k z^k$. Therefore, we might consider the joint observation of z_{t_k} and $\hat{f}^k(\theta) \equiv f(\hat{x}_{t_k}, \theta)$ at each time point and attempt to solve the decision tree defined by the policy problem that contains both observations at each chance node. We address this last idea later; here we explore the arguably more natural situation where only one hat function is considered at this point.

6.3.1 Just one hat function

We describe the Sequential Emulation algorithm in the case where z_{t_0} is to be used to define a hat function, no other hat functions are being considered, and we must solve the policy problem of section 3.1.3. In the original Sequential Emulation algorithm we must sample from distributions $p(y|z^m, \theta^m)$ and $p(z_{t_k}|z^{k-1}, \theta^{k-1})$ for $k = 1, \ldots, m$. If using Bayes Linear forecasting to characterize these distributions, then we compute either $E_{z^m}[y; \theta^m]$ and $Var_{z^m}[y; \theta^m]$, or $E_{z^{k-1}}\left[z_{t_k}; \theta^{k-1}\right]$ and $Var_{z^{k-1}}\left[z_{t_k}; \theta^{k-1}\right]$ using the forecasting methods of section 2.3.2, and then make some distributional assumption that uses the appropriate adjusted mean and variance to characterize a probability distribution.

The original Sequential Emulation algorithm requires multi-level emulation of expectations taken with respect to each of these distributions. For both coarse and accurate methods of calculating these expectations, we are required, as part of preliminary steps to be performed, to decide what our forecasting methods will be. For any method that we choose to be Bayes Linear, we may now use a calibrated forecast based on $\hat{f}(\theta)$. There may be situations where we do not use Bayes Linear calibrated forecasts, even though we have performed a hat spin based on z_{t_0} and can compute $\hat{f}(\theta)$. A particular example would be when, even after spin up, $\hat{f}(\theta)$ takes a long time to compute. We may then choose to allow coarse forecasting methods to be the same as in section 2.3.2, and only use calibrated forecasts for accurate calculations.

Supposing that we can compute calibrated forecasts for each of the quantities required to characterize $p(y|z^m, \theta^m)$ and $p(z_{t_k}|z^{k-1})$ for $k = 1, \ldots, m$, then, the Sequential Emulation algorithm that includes the possibility of using a hat function differs from the original Sequential Emulation algorithm only at step P3. Step P3 requires that we select both accurate and fast forecasting calculations, where 'fast' forecasting calculations will be Bayes Linear forecasts. P3 changes here by allowing us to choose Bayes Linear calibrated forecasts for either fast or accurate forecasting methods or both. If Bayes Linear calibrated forecasts are chosen, we must compute $\hat{x}$, spin the model up, and emulate the required moments of $\hat{f}(\theta)$ using the methods described in the previous section. Once this is done, we characterize the required

distributions for Sequential Emulation using Bayes Linear calibrated forecasts in the following way.

Firstly, the distribution for $p(y|z^m, \theta^m, \hat{f}(\theta))$ is characterized using $E_{z^m,\hat{f}}[y;\theta^m]$ and $Var_{z^m,\hat{f}}[y;\theta^m]$ with

$$E_{z^m,\hat{f}}[y;\theta^m] = E[y] + Cov\left[y, (z^m, \hat{f}(\theta))\right] Var\left[z^m, \hat{f}(\theta)\right]^{-1} \left((z^m, \hat{f}(\theta)) - E\left[z^m, \hat{f}(\theta)\right]\right)$$

and

$$Var_{z^m,\hat{f}}[y;\theta^m] = Var[y] - Cov\left[y, (z^m, \hat{f}(\theta))\right] Var\left[z^m, \hat{f}(\theta)\right]^{-1} Cov\left[(z^m, \hat{f}(\theta)), y\right].$$

The moments on the collection $\{\hat{f}(\theta), z^m, y\}$ are emulated beforehand as part of P3. For $k = 1, \ldots, m$, the distributions $p(z_{t_k}|z^{k-1}, \theta^{k-1}, \hat{f}(\theta^{k-1}))$ are characterized similarly using $E_{z^{k-1},\hat{f}}\left[z_{t_k};\theta^{k-1}\right]$ and $Var_{z^{k-1},\hat{f}}\left[z_{t_k};\theta^{k-1}\right]$.

To be absolutely clear: if we are able to use Bayes Linear calibrated forecasts based on the hat function defined by z_{t_0} in our Sequential Emulation of the decision tree, then the Sequential Emulation algorithm changes only at step P3, where we replace P3 with $\hat{\text{P3}}$.

$\hat{\text{P3}}$

Select both 'accurate' and 'fast' forecasting methods When performing the Sequential Emulation algorithm we will require $\frac{1}{2}(m+1)(m+2)$ multi-level emulations. Each of these is an expectation with respect to a probability distribution for future states or observations of the complex system. These distributions are either derived exactly from our forecasting methods, or we use Bayes Linear forecasts to characterize these distributions. The observation z_{t_0} defines a hat function that we may use to provide calibrated forecasts for either fast or accurate methods. We first decide whether or not Bayes Linear calibrated decision-dependent forecasts will be used for coarse versions, accurate versions or both. We then perform the hat spin and emulate the hat function moments in the way described in section 6.2. How accurate our emulation of the hat function moments is and how carefully we perform the required numerical integrations with respect to x^* is a decision that must be taken at this point, and will depend on whether we are using the forecasts in

coarse or accurate evaluations of our expectations, and at what stage our analysis is. For example, if we have pruned our tree and refocussed many times, we may be prepared to use all of our remaining time to obtain the most accurate calibrated forecasts we can. On the other hand, if this is our first Sequential Emulation of the decision tree and we intend to prune and refocus many times, we may choose not to use calibrated forecasting at all on the first sweep.

6.3.2 Future hat functions

It is natural to include the hat function that we may evaluate now for any θ into our Sequential Emulation framework because we have already computed $\hat{x}$ and spun the model up. As discussed in section 6.3.1, the Sequential Emulation algorithm barely changes and Bayes Linear calibrated forecasts are used as part of either coarse or accurate calculations, or both, as the user sees fit.

If Bayes Linear calibrated forecasting will be our future method of choice when assessing the impact of new system observations, however, by following this version of the algorithm we are not emulating our future behaviour. At some future time point, t_k, we observe data from the complex system z_{t_k}. We could then, in theory, define

$$\begin{aligned}\hat{x}^k &= v^k + W^k z^k \\ &= E_{z^k}\left[x^*\right]\end{aligned}$$

and

$$\hat{f}^k(\theta) \equiv f(\hat{x}^k, \theta),$$

so that $\hat{f}^k(\theta)$ is the hat function that, if we spun the model up at $\hat{x}^k$ having observed z_{t_k}, we would be able to evaluate at time t_k. Knowing that each observation of the system that we may make in the future defines a new hat function alters our decision tree in the following way: after each observation z_{t_k}, we must decide whether or not to spin the model up at $\hat{x}^k$, and if we do we may observe $\hat{f}^k(\theta)$ for any θ before making our next decision.

An example of this decision tree where we only have one intervention point is shown in figure 6.1. Providing policy support for such a decision tree, including the

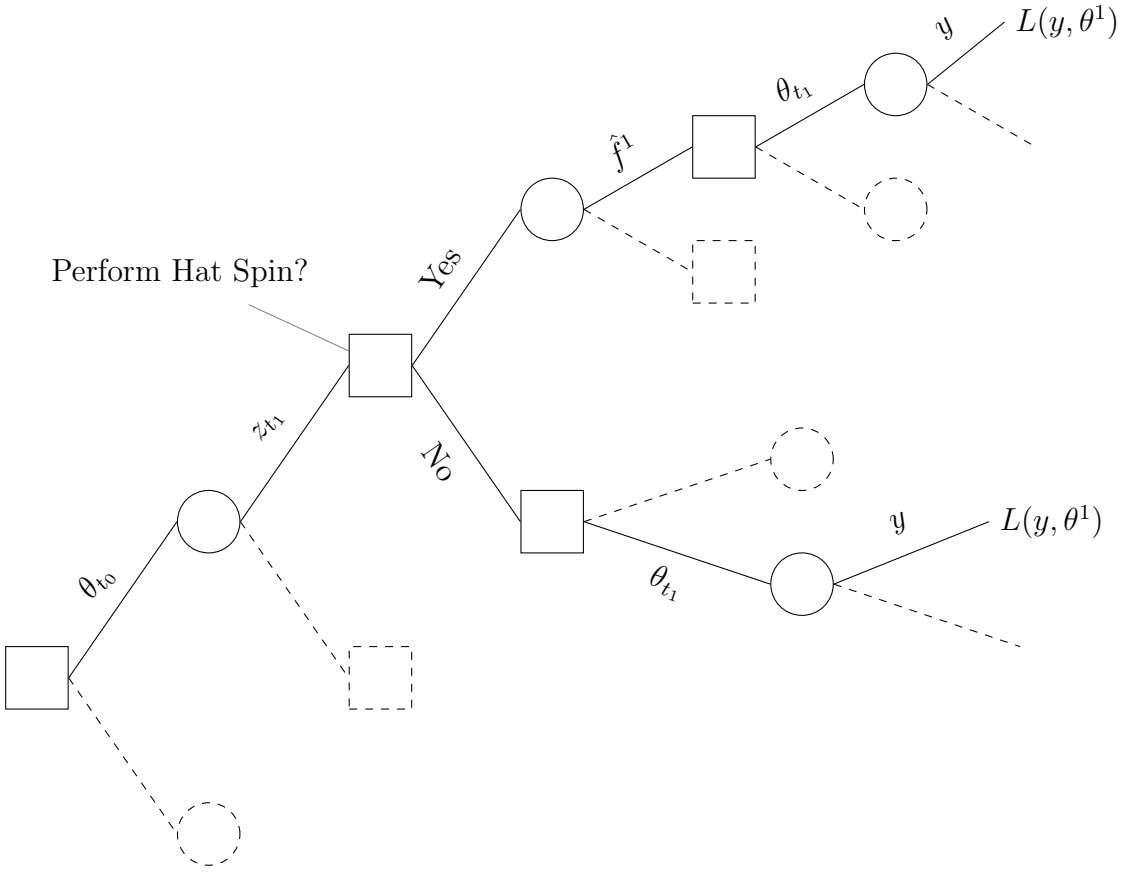

Figure 6.1: An example of the decision tree defined by the policy problem with one intervention point that includes the option of performing the hat spin and evaluating the hat function defined by any z_{t_1}. Although the tree marks the observation of $\hat{f}^1$ if we choose to perform the hat spin, we argue in the main text that this observation is a surface in θ_{t_1}. This surface is not observed in its entirety having performed the hat spin, but we are able to evaluate it for many θ_{t_1} by evaluating the hat function.

most general case where we have m potential hat functions, is beyond the scope of this thesis. However, we include some ideas for approaching the problem via an adapted Sequential Emulation here, for the case where $m = 1$.

In order to emulate the decision tree in figure 6.1, we must be able to sample from the distribution $p(y|\hat{f}^0, \hat{f}^1, \theta^1, z^1)$, where $\hat{f}^0$ is defined to be the hat function we can evaluate now. We would do this by computing $E_{z^1,\hat{f}^1,\hat{f}^0}[y; \theta^1]$ and $Var_{z^1,\hat{f}^1,\hat{f}^0}[y; \theta^1]$ and characterizing a distribution for y using its adjusted moments in the same way as

seen before. In order to compute these forecasts we would require the joint moments of the collection $\{\hat{f}^0(\theta), \hat{f}^1(\theta), y, z^1\}$, which would be obtained using the computer algebra package methods to build an emulator for them in the same way that we did for one hat function.

This emulator could be built before addressing the decision problem and, if it is feasible to emulate the moments of one hat function, there is no reason that emulating the joint moments of two or more hat functions would not be feasible. What would make this decision tree difficult to emulate is the way in which $\hat{f}^1(\theta_{t_1})$ informs our choice of θ_{t_1}. If we assume that, having spun the model up at $\hat{x}^1$, we can evaluate $\hat{f}^1(\theta_{t_1})$ for any θ_{t_1} cheaply to inform our downstream decision making, it is difficult to see how $\hat{f}^1(\theta_{t_1})$ is 'observed' before we take our downstream decision. Put another way, each branch of the decision tree leading up to θ_{t_1} contains a random function of θ_{t_1}. Hence, having observed z_{t_1} and chosen to perform the new hat spin, the 'observation' of $\hat{f}^1(\theta_{t_1})$ is a realisation of a stochastic function of θ_{t_1} that may now be evaluated. We, in fact, observe a surface in θ_{t_1} before choosing a downstream decision.

The difficulties that must be overcome if we are to apply our Sequential Emulation techniques to this decision tree are, then: how to sample random functions of θ_{t_1} from a distribution $p(\hat{f}^1(\theta_{t_1})|\hat{f}^0(\theta^1), \theta_{t_0}, z^1)$ so that, once sampled, we can then evaluate the sampled function for many different θ_{t_1}; and how to describe 'observed' functions of θ_{t_1} in a way that allows us to determine downstream strategy λ_{t_1} as a function of these observations.

To make this clearer, suppose we have chosen a θ_{t_0}, observed z_{t_1} and observed the surface $\hat{f}^1(\theta_{t_1})$. Our downstream strategy, if we were using Sequential Emulation, would be defined by

$$\lambda_{t_1}(\theta_{t_0}, z^1, \hat{f}^1(\theta_{t_1})) = arg \min_{\theta_{t_1}} \{\tilde{E}\left[\int L(y,\theta)p(y|\hat{f}^0, \hat{f}^1, \theta^1, z^1)dy\right]\}. \tag{6.13}$$

Irrespective of how we might build such an emulator, we must have a way of parametrizing observations of $\hat{f}^1$ that define a function of θ_{t_1} that we are able to evaluate quickly. Without this we could not see how optimal strategy changed with the inclusion of a second hat function when compared to the case where we do not spin the model up again. If we cannot find a way of parametrizing $\hat{f}^1(\theta_{t_1})$ so that

an observation of $\hat{f}^1(\theta_{t_1})$ defines a curve in θ_{t_1}, the Sequential Emulation methods, which rely on defining downstream strategies as functions of quantities already observed and decisions already made, must either be radically changed or abandoned altogether. To be clear, if we wish to employ Sequential Emulation methods for this decision problem, an observation of $\hat{f}^1(\theta_{t_1})$ must fix a curve in θ_{t_1} that has no stochastic properties and can be readily evaluated for any θ_{t_1}.

For any given branch of the decision tree, θ_{t_0} and z_{t_1} are fixed. Therefore, $\hat{x}^1$ is fixed and we have

$$\hat{f}^1(\theta_{t_1}) = \beta_{ij} g_j(v^1 + W^1 z^1, (\theta_{t_0}, \theta_{t_1})) + u_i(\hat{x}^1, (\theta_{t_0}, \theta_{t_1})).$$

Using a computer algebra package or otherwise, we may write this as

$$\hat{f}^1(\theta_{t_1}) = \alpha_{ij}(\theta_{t_0}, z^1, \beta)\zeta_j(\theta_{t_0}, z^1, \theta_{t_1}) + u_i(\hat{x}^1, (\theta_{t_0}, \theta_{t_1})), \tag{6.14}$$

where $\alpha_{ij}(\theta_{t_0}, z^1, \beta)$ is a matrix whose entries are known functions of θ_{t_0}, z^1 and β, and $\zeta_j(\theta_{t_0}, z^1, \theta_{t_1})$ is a vector of functions that are either 1 or contain θ_{t_1}.

If we ignored $u(\hat{x}^1, (\theta_{t_0}, \theta_{t_1}))$ for a moment, then observing β fixes α and, without $u(\hat{x}^1, (\theta_{t_0}, \theta_{t_1}))$, would determine a curve in θ_{t_1} that could be readily evaluated. As $u(\cdot, \cdot)$ is a random process, however, it is difficult to determine a way in which an observation of $\hat{f}^1(\theta_{t_1})$ defines a deterministic curve in θ_{t_1}. One idea is to expand $u(\hat{x}^1, (\theta_{t_0,\theta_{t_1}}))$ as a Taylor series in θ_{t_1} and approximate $u(\hat{x}^1, (\theta_{t_0}, \theta_{t_1}))$ by a higher order polynomial derived using the expansion. For example, suppose we let

$$u_i(\hat{x}^1, (\theta_{t_0}, \theta_{t_1})) \approx \gamma_{i0} + \gamma_{i1}\theta_{t_1} + \gamma_{i2}\theta_{t_1}^2 + \ldots + \gamma_{in}\theta_{t_1}^n. \tag{6.15}$$

Then an observation of a curve $\hat{f}^1(\theta_{t_1})$ is defined by observation of α and γ. Our time t_1 strategy λ_{t_1} would then be a function of θ_{t_0}, z^1, α and γ. Given any α, γ and θ_{t_1} we could compute $\hat{f}^1(\theta_{t_1})$ using (6.14) and (6.15) and, therefore, sample from $p(y|\hat{f}^0, \hat{f}^1, \theta^1, z^1)$ and compute expected loss. In order to be able to perform a Sequential Emulation within this framework, we would need, now, to specify how to sample from $p(\alpha, \gamma|\hat{f}^0, z^1, \theta_{t_0})$. Whilst there may be many ways of doing this, we suggest the following approach. If we can use $\hat{f}^0(\theta_{t_0})$ and z_{t_1} to appropriately condition $\{\beta, u(\cdot)\}$, then we could sample a value of β to obtain a sample value for

α, condition $u(\cdot)|\hat{f}^0(\theta_{t_0}), z_{t_1}$ by this sampled value of β, and then sample a very large number of values of this random field over some design in θ_{t_1}. This large sample (as long as it was bigger than the order of the polynomial approximation in (6.15)) could be used to fit values of γ, thus generating a sample from $p(\alpha, \gamma|\hat{f}^0, z^1, \theta_{t_0})$. One way in which we might achieve the required conditioning of β and $u(\cdot)$ by $\hat{f}^0(\theta_{t_0})$ and z_{t_1} is to use approximate Bayes computing.

We can evaluate $\hat{f}^0(\theta_{t_0})$, for any θ_{t_1}, using the hat spin that has already been performed. Suppose we have conditioned β and $u(x, \theta)$ on runs F already. Then we may use approximate Bayes computing in the following way. We generate an x^*, β, $u(x^*, \theta_{t_0})$, η and e_{t_1} from the appropriate distributions to compute a candidate value of z_{t_1}. We do this in the same way as for the large sampling experiments we advocated in section 6.1.1. If the candidate is within a certain pre-chosen neighbourhood of our z_{t_1}, then we use the sampled values of β and $u(x^*, \theta_{t_0})$ to define a model run $\tilde{f}(x^*, \theta)$. We then adjust $\{\beta, u(\cdot)\}$ by $\hat{f}^0(\theta_{t_0})$ and $\tilde{f}(x^*, \theta)$, and judge that these new beliefs on that random field are appropriately conditioned on our values of z_{t_1} and $\hat{f}^0(\theta_{t_0})$. If the candidate is outside that neighbourhood, we generate a new candidate.

The types of problems we must overcome if we were to adopt a Sequential Emulation approach for providing insight into the decision tree are discussed above. If, however, we are prepared to take a more pragmatic but, perhaps, less accurate approach in order assess the impact of future hat functions on our expected losses, then we may use certain simplifications to help us. One idea is to first assess an order of magnitude reduction in our variance for y attributable to future hat functions. This might be achieved, for example, using some form of sampling experiment. Having performed this assessment, we could then explore the impact on the decision tree of reducing the forecast variance by the amounts given. Such an approach is similar, in spirit, to the approach we adopted for handling future model observations on as yet unbuilt models in chapter 5. In that case we found it impossible to deal with the full decision tree that contained all possible designs of future model runs, and replaced the observation of runs by the observation of the adjusted coefficients on our emulator and an overall reduction in forecast uncertainty experienced having

built and performed runs on a new model.

We have presented a number of ideas regarding how our Sequential Emulation methodology might be adapted to help provide policy support for problems where we know we may wish to define a new hat function or several new hat functions in the future. It is unclear whether or not Sequential Emulation would be appropriate in such decision problems. The goal in evaluating a new hat function is to resolve more of our local uncertainty around x^*, having already evaluated the hat function defined by observations of the system obtained before choosing θ_{t_0}. When building emulators for deterministic functions, we introduce uncertainty into our description of the function. If a new, future, hat function is only there to reduce an already small amount of uncertainty, will any benefits of that reduction in uncertainty show up in our analysis or will they be lost in the new emulator uncertainty that we would be introducing? If the ideas we have discussed or others were to be used to provide an emulation of this decision problem, this question must be seriously addressed. It may be that there is a better way to provide insight into this decision problem and we leave this as an open research question.

Chapter 7

Concluding remarks

In this thesis we set out to investigate how some of the sophisticated methods available for making inference regarding future states of complex systems using computer models, might be used in order to provide decision support in large scale policy problems. As part of this investigation, it was our aim to generalize some of the Bayes Linear forecasting technologies in the computer experiment literature to the case where we could make decisions in order to affect future states of a complex system, and where these decisions are inputs to our model.

We began our investigation in chapter 2 by reviewing the current methods for emulating computer models in order to make inference about complex systems. We described current Bayes Linear forecasting methods within the computer experiment literature and generalized them by defining Bayes Linear decision-dependent forecasts. We remarked that the amount of numerical integration required to provide just one forecast may render the idea of exploring how our beliefs change as we move through the decision space infeasible. In section 2.3.3 we demonstrated how separability, where it exists in our emulator, may remove any dependence on θ from our forecast integrals. In section 2.3.4 we defined fast forecasts and presented a number of different ideas that exploit various potential features of the correlation function of our emulator residual, in order to produce computationally efficient approximations to our forecasts.

Depending on the nature of the problem, the speed at which decision-dependent forecasts may be obtained could be crucial. Whilst we believe that the fast forecast-

ing methods we have proposed are useful, we have left plenty of scope for further work. In particular, the question of how much of the accuracy of the forecast we are prepared to trade off for computational efficiency, and how one quantifies those two ideas in the context of the decision problem, is an area which remains to be explored.

In chapter 3 we described the policy problem where future system observations could be made and policy strategies adapted accordingly. We discussed how current methods in Integrated Assessment modelling provide decision support for these problems and identified a number of drawbacks within that approach. The principle limitations were that there is no formal treatment of the uncertainty regarding the discrepancy between the real world and the computer model, and that in order to retain tractibility the most powerful computer models available for a particular complex system could not be used. We presented an alternative approach that used current methods for making inference about complex systems with computer models in order to combine information from the most powerful simulator available and real-world observations of the complex system in our analysis of the decision problem. The decision tree that we defined could not be solved exactly. We, therefore, developed a method which we called Sequential Emulation, designed to emulate our decision tree in order to provide decision support. The method relied heavily on Bayes Linear multi-level emulation of expected losses, with coarse evaluations derived using Bayes Linear decision-dependent fast forecasts.

In performing a Sequential Emulation of a decision tree, we obtain an emulator for an upper bound on our expected loss for making policy today. We also obtain policy adaptation strategies for each of the times at which we intend to observe the system and revise policy. These strategies are in the form of emulators that are functions of the decisions already made and the observations of the system that we have. In section 3.7 we described some of the many ways in which these results can be used to provide policy support. These included pruning and refocussing, sensitivity analysis and risk profiling. We provided examples of these policy support ideas in practice in chapter 4. The policy support ideas we presented did not constitute an exhaustive list of the possible applications of the results of a Sequential Emulation.

We also did not provide any concrete methodology for deciding how and where to prune our decision space based on the output of a Sequential Emulation.

There are many possible avenues for further work opened up by our Sequential Emulation methodology. How one visualizes each of the downstream strategies and assesses their performance; how one decides where to prune the decision space, and when to stop pruning and refocussing and make the final decision; and how one assesses the quality of the emulator for the upper bound are all examples of potential future directions. Our methods were developed under the assumption that the loss function is known. The extension of our methods to the case where we have a loss model with many uncertain parameters, and whose relation to our true preferences is uncertain, is a natural area for further work. We provided early discussion regarding this in section 3.8.

Sequential Emulation assumes that we would like to do a full backwards induction on our decision problem, and then aims to emulate this process. There may be more advanced decision theoretic methods for solving infinite decision trees, such as those generated by the policy problems we have discussed, in the literature. It is likely that, because we must use expensive forecasting methods to derive the probabilities on our tree, these methods would still be computationally challenging. In this case, the idea of emulating the process of solving the decision tree, in the same way that our Sequential Emulation mimics backwards induction, may be a way forward. Further work in this area may consider different methods for solving decision trees and applying the Sequential Emulation ideas within these frameworks in order to provide decision support for policy problems.

In chapter 5 we addressed the important issue of evolving computer models and how we may use our current model to provide decision support when it is known that we may be able to observe runs on improved models in the future. We generalized some of the reification techniques for computer models introduced by Goldstein and Rougier in [40] to the case where we have decisions as inputs to our model and where those decisions affected future states of the complex system.

We used structural reification to develop a treatment of observations of output from future models that was tractible and that allowed us to derive decision-

dependent forecasts that were appropriately conditioned on these observations. We called these reified decision-dependent forecasts and decision-dependent observation forecasts. We showed how, within the framework that we had established, we were able to use Reified Sequential Emulation to provide policy support for problems where the computer models are continually evolving and how we are able to consider certain research investment decisions concurrently. In particular, the decision of whether or not to invest in a certain improved model and how much to run that model, can be included as part of the policy problem. We demonstrated this using an extension to the example we presented in chapter 4.

There are a number of potential avenues for further work in this area. Our methods treat future model observations as observations of the adjusted expectation of the emulator coefficients for that model. If our uncertainty on the residuals for future models is large, further modelling may be required in order to perform meaningful policy support.

There may be a significant number of potential policy support tools that could be derived from the results of a Reified Sequential Emulation, particularly related to visualization of downstream policy and model build strategies, as well as methods specifically focussed on the research investment questions posed by the ability to build and run improved simulators. We gave a short discussion of this matter in section 5.9.2. Another extension to this methodology could involve providing policy support when not only the computer models for the complex system are evolving, but when we have a loss model which is continually being improved upon as well.

In chapter 6 we moved away from the case where the models are improving, and focussed on generalizing the Bayes Linear calibrated forecasting methodology of Goldstein and Rougier [39] to the case where our computer model had decision inputs, and where those decisions affected future states of the complex system. We showed how observations of the complex system defined a hat function, and described how, for models with a spin up property, this hat function could provide very useful information around the best input. We argued that the hat function moments would need to be emulated if we were to effectively exploit the spin up property, and showed how a computer algebra package might be employed to limit

the amount of emulator uncertainty introduced into our forecasts.

In section 6.3 we showed how the Sequential Emulation algorithm could be adapted to allow the use of Bayes Linear calibrated forecasts when evaluating expected losses. We did this for the case where only the hat function, defined by z_{t_0}, was considered. Finally, we pointed out the issues involved in considering the more general case where future observations of the system are allowed to define new hat functions, and we may decide whether or not to perform the new hat spin. We offered insight into how Sequential Emulation methods might be adapted to cater for these problems. It may be, however, that future work in this area would have to move away from Sequential Emulation, as was discussed at the end of the chapter.

The Bayesian analysis of computer experiments and how they may be used to make inference regarding complex systems, is a subject in its infancy. One of the key reasons for building a computer model of a complex system is to inform policy judgements. The work we have done has generalized some of the modern methods of forecasting future states of complex systems using computer models, to the case where decisions, parametrized as inputs to the model, can be taken in order to influence those future states. We showed how formal treatment of the model and the system defined an infinite decision tree, and our Sequential Emulation technology represents a useful method for providing policy support in many different contexts. We believe that these methods represent first steps towards the goal of being able to choose the optimal course of action in real world policy problems, when computer models are used to inform scientific judgements regarding complex systems.

Bibliography

[1] United nations framework convention on climate change. http://unfccc.int/2860.php.

[2] Mucm toolkit. http://mucm.aston.ac.uk/MUCM/MUCMToolkit/index.php?page=AltCoreDesign.html, 2009.

[3] Jian An and Art Owen. Quasi-regression. *Journal of Computing*, 17:588–607, 2001.

[4] J. Baehr, K. Keller, and J. Marotzke. Detecting potential changes in the meridional overturning circulation at 26°n in the atlantic. *Climatic Change*, 91(1-2):11–27, 2008.

[5] S. C. Bankes. Tools and techniques for developing policies for complex and uncertain systems. *PNAS*, 99, May 2002. Colloquium Paper.

[6] M. S. Bartlett. *An Introduction to Stochastic Processes, with special reference to methods and applications*. Cambridge University Press, Cambridge, London, New York, Melbourne, 1978.

[7] L. S. Bastos and A. O'Hagan. Diagnostics for gaussian process emulators. Technical report, University of Sheffield, 2008.

[8] R. A. Bates, R. J. Buck, E. Riccomagno, and H. P. Wynn. Experimental design and observation for large systems. *Journal of the Royal Statistical Society, Series B*, 58(1):77–94, 1996.

[9] M. J. Bayarri, J. O. Berger, D. Higdon, M. C. Kennedy, A. Kottas, R. Paulo, J. Sacks, J. A. Cafeo, J. Cavendish, C. H. Lin, and J. Tu. A framework for validation of computer models. *Technometrics*, 49(2):138–154, 2007.

[10] J. Berger. The case for objective bayesian analysis. *Bayesian Analysis*, 1(3):385–402, 2006. Including comments and rejoindre.

[11] G. Blatman, B. Sudret, and M. Berveiller. Quasi-random numbers in stochastic finite element analysis. *Mécanique & Industries*, 8:289–297, 2007.

[12] R. G. Bower, A. J. Benson, R. Malbon, J. C. Helly, C. S. Frenk, C. M. Baugh, S. Cole, and C. G. Lacey. The broken hierarchy of galaxy formation. *Monthly Notices of the Royal Astronomical Society*, 370:645–655, 2006.

[13] M. Brugnach, A. Tagg, F. Keil, and W. J. de Lange. Uncertainty matters: Computer models at the science policy interface. *Water Resource Management*, 21:1075–1090, 2007.

[14] David V. Budescu, Robert Lempert, Stephen Broomell, and Klaus Keller. Aided and unaided decision making with imprecise probabilities. *Risk Analysis*, 2009.

[15] E. A. Casman, M. G. Morgan, and H. Dowlatabadi. Mixed levels of uncertainty in complex policy models. *Risk Analysis*, 19(1), 1999.

[16] P. Challenor and B. Marsh. First steps towards the estimation of the probability of thermohaline collapse. In *Avoiding dangerous climate change*, pages 55–63. Cambridge University Press, 2006. Chapter 7.

[17] P. S. Craig, M. Goldstein, Rougier J. C., and A. H. Seheult. Bayesian forecasting for complex systems using computer simulators. *Journal of the American Statistical Association*, 96(454):717–729, 2001.

[18] P. S. Craig, M. Goldstein, A. H. Seheult, and J. A. Smith. Bayes linear strategies for matching hydrocarbon reservoir history. In J. M. Bernado, J. O. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics 5*, pages 69–95. Oxford University Press, 1996.

[19] P. S. Craig, M. Goldstein, A. H. Seheult, and J. A. Smith. Pressure matching for hydrocarbon reservoirs: A case study in the use of bayes linear strategies for large computer experiments. In C. Gatsonis, J. S. Hodges, R. E. Kass, R. McCulloch, P. Rossi, and N. D. Singpurwalla, editors, *Case Studies in Bayesian Statistics*, volume III, pages 36–93. New York: Springer-Verlag, 1997.

[20] N. A. C. Cressie. *Statistics For Spatial Data.* John Wiley and sons, Inc, 1993.

[21] J. A. Cumming and M. Goldstein. Bayes linear uncertainty analysis for oil reservoirs based on multiscale computer experiments. In A. O'Hagan; M. West, editor, *In submission. — to: The Handbook of Bayesian Analysis.* 2009.

[22] J. A. Cumming and M. Goldstein. Small sample designs for complex high-dimensional models based on fast approximations. *In Submission. — to: Technometriks*, 2009.

[23] J. A. Cumming and D. A. Wooff. Dimension reduction via principal variables. *Computational statistics and data analysis*, 52:550–565, 2007.

[24] C. Currin, T. Mitchell, M. Morris, and D. Ylvisaker. Bayesian prediction of deterministic functions, with applications to the design and analysis of computer experiments. *Journal of the American Statistical Association*, 86:953–963, 1991.

[25] Philip J. Davis and Philip Rabinowitz. *Methods of Numerical Integration.* Academic Press Inc., second edition, 1984.

[26] Morris H. DeGroot. *Optimal statistical decisions.* McGraw-Hill Inc, 1970.

[27] P. J. Diggle and P. J. Ribeiro Jr. *Model-based Geostatistics.* Springer Science and Business Media, LLC, 2007.

[28] L. Drouet, N. R. Edwards, and A. Haurie. Coupling climate and economic models in a cost-benefit framework: A convex optimisation approach. *Environmental Modelling and Assessment*, 2:101 – 114, 2006.

[29] N. R. Edwards and R. Marsh. Uncertainties due to transport-parameter sensitivity in an efficient 3-d ocean-climate model. *Climate Dynamics*, (4):415–433, 2005.

[30] Gwynne Evans. *Practical Numerical Integration.* John Wiley and Sons Ltd, 1993.

[31] A. D. Fernandes and W. R. Atchley. Gaussian quadrature formulae for arbitrary positive measures. *Evolutionary Bioinformatics Online*, 2:251–259, 2006.

[32] B. De Finetti. *Theory of Probability.* John Wiley and Sons, Ltd, 1974.

[33] A. I. J. Forrester and A. J. Keane. Recent advances in surrogate-based optimization. *Progress in Aerospace Sciences*, 45:50–79, 2009.

[34] A. E. Gelfand and D. K. Dey. Bayesian model choice: Asymptotics and exact calculations. *Journal of the Royal Statistical Society, Series B*, 1994.

[35] A. A. Giunta, M. S. Eldred, T. G. Trucano, and S. F. Wojtkiewicz jr. Optimization under uncertainty methods for computational shock physics applications. In *43rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural dynamics, and Materials Conference*, Denver, CO, AIAA, 2002.

[36] M. Goldstein. Prior inferences for posterior judgements. In J. van Bentham M. L. D. Chiara; K. Doets; D. Mundici, editor, *Structures and Norms in Science. Volume Two of the Tenth International Congress of Logic, Methodology and Philosophy of Science, Florence*, pages 55–71, 1997.

[37] M. Goldstein. Subjective bayesian analysis: Principles and practice. *Bayesian Analysis*, 1(3):403–420, 2006. Including comments and rejoindre.

[38] M. Goldstein and J. C. Rougier. Probabilistic formulations for transferring inferences from mathematical models to physical systems. *SIAM Journal on Scientific Computing*, 26(2):467–487, 2004.

[39] M. Goldstein and J. C. Rougier. Bayes linear calibrated prediction for complex systems. *Journal of the American Statistical Association*, 2006.

[40] M. Goldstein and J. C. Rougier. Reified bayesian modelling and inference for physical systems. *Journal of statistical planning and inference*, 2009.

[41] M. Goldstein and I. Vernon. Bayes linear analysis of imprecision in computer models, with application to understanding the universe. In *6th International Symposium on Imprecise Probability: Theories and Applications*. 2009.

[42] M Goldstein and D. Wooff. *Bayes Linear Statistics Theory and Methods*. John Wiley and Sons Ltd, 2007.

[43] R. Haylock and A. O'Hagan. On inference for outputs of computationally expensive algorithms with uncertainty on the inputs. In J. M. Bernado, J. O. Berger, A. P. Dawid, and A. F. M. Smith, editors, *Bayesian Statistics 5*, pages 629–637. Oxford University Press, 1996.

[44] D. Higdon, M. Kennedy, J. C. Cavendish, J. A. Cafeo, and R. D. Ryne. Combining field data and computer simulations for calibration and prediction. *SIAM Journal on Scientific Computing*, 26(2):448–466, 2004.

[45] J. T. Houghton, Y. Ding, D. J. Griggs, M. Noguer, P. J. van der Linden, and D. Xiaosu, editors. *Contribution of working group 1 to the third assessment report of the Intergovernmental Panel on Climate Change*. Cambridge Univeristy Press, 2001.

[46] J. T. Houghton, L. G. Meira Filho, B. A. Callender, N. Harris, A. Kattenberg, and K. Maskell, editors. *Contribution of working group 1 to the second assessment of the Intergovernmental Panel on Climate Change*. Cambridge University Press, 1995.

[47] D. Huang, T. T. Allen, W. I. Notz, and N. Zheng. Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of Global Optimization*, 34:441–466, 2006.

[48] Technical Support Unit IPCC-WG3. Sres final data (version 1.1). `http://sres.ciesin.org/final_data.html`, July 2000.

[49] Y. A. Izrael and S. M. Semenov. Critical levels of greenhouse gases, stabilization scenarios, and implications for the global decisions. In *Avoiding dangerous climate change*, chapter 9, pages 73–79. 2006.

[50] D. R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383, 2001.

[51] D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998.

[52] P. D. Jones, M. New, D. E. Parker, S. Martin, and I. G. Rigor. Surface air temperature and it's changes over the past 150 years. *Reviews of Geophysics*, 37(2):173–199, 1999.

[53] V. R. Joseph, Y. Hung, and A. Sudjianto. Blind kriging: A new method for developing metamodels. *Journal of Mechanical Design*, 130(3), 2008.

[54] A. Kann and J. P. Weyant. Approaches for performing uncertainty analysis in large-scale energy/economic policy models. *Environmental modelling and assessment*, 2000.

[55] K. Keller, M. Hall, S. Kim, D. F. Bradford, and M. Oppenheimer. Avoiding dangerous anthropogenic interference with the climate system. *Climatic Change*, 2005.

[56] K. Keller, D. McInerney, and D. F. Bradford. Carbon dioxide sequestration: how much and when? *Climatic Change*, 2008.

[57] M. C. Kennedy and A. O'Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1), 2000.

[58] M. C. Kennedy and A. O'Hagan. Bayesian calibration of computer models. *Journal of the Royal Statistical Society. Series B*, 2001.

[59] R. J. T. Klein, E. L. F. Schipper, and S. Dessai. Integration mitigation and adaptation into climate and development policy: three research questions. *Environmental Science and Policy*, 8:579–588, 2005.

[60] J. R. Koehler and A. B. Owen. Computer experiments. In S. Ghosh and C. R. Rao, editors, *Handbook of Statistics 13: Design and Analysis of Experiments*, pages 261–308. North-Holland: Amsterdam, 1996.

[61] S. J. Leary, A. Bhaskar, and A. J. Keane. A knowledge-based approach to response surface modelling in multifidelity optimization. *Journal of Global Optimization*, 26:297–319, 2003.

[62] M. Leimbach, N. Bauer, L. Baumstark, and O. Edenhofer. Mitigation costs in a globalized world: climate policy analysis with remind-r. *Environmental modeling and assessment*, to appear in 2010.

[63] M. Leimbach and C. Jaeger. A modular approach to integrated assessment modeling. *Environmental modeling and assessment*, 9(4):207–220, 2005.

[64] R. J. Lempert. A new decision sciences for complex systems. *PNAS*, 99, May 2002. Colloquium Paper.

[65] R.J. Lempert, M. E. Schlesinger, and S. C. Bankes. When we don't know the costs or the benefits: adaptive strategies for abating climate change. *Climatic Change*, 33:235–274, 1996.

[66] R.J. Lempert, M. E. Schlesinger, S. C. Bankes, and N. G. Andronova. The impacts of climate variability on near-term policy choices and the value of information. *Climate Change*, 2000.

[67] M. Liefendahl and R. Stocki. A study on algorithms for optimization of latin hypercubes. *Journal of statistical planning and inference*, 136:3231–3247, 2006.

[68] I. Linkov and D. Burmistrov. Model uncertainty and choices made by modelers: Lessons learned from the international atomic energy agency model intercomparisons. *Risk Analysis*, 23(6), 2003.

[69] D. M. Maniyar, D. Conford, and A. Boukouvalas. Dimensionality reduction in the emulator setting. Technical report, NCRG Aston University, 2007.

[70] Maplesoft. Maple 12, 2008. Maplesoft is a division of Waterloo Maple Inc.

[71] K. V. Mardia, J. T. Kent, and J. M. Bibby. *Multivariate Analysis*. Academic Press, Harcourt Brace and Company, Publishers, 1979.

[72] J. D. Martin and T. W. Simpson. On the use of kriging models to approximate deterministic computer codes. *AIAA Journal*, 43:853–863, 2005.

[73] D. McInerey and K. Keller. Economically optimal risk reduction strategies in the face of uncertain climate thresholds. *Climatic Change*, 2006.

[74] J. H. Mercer. West antarctic ice sheet and co2 greenhouse effect: a threat of disaster. *Nature*, 1978.

[75] M. G. Morgan and D. W. Keith. Subjective judgements by climate experts. *Environmental Science and Technology*, 29(10), 1995.

[76] M. G. Morgan, L. F. Pitelka, and E. Shevliakova. Elicitation of expert judgements of climate change impacts on forest ecosystems. *Climatic Change*, 49:279–307, 2001.

[77] M. D. Morris and T. J. Mitchell. Exploratory designs for computational experiments. *Journal of statistical planning and inference*, 43:381–402, 1995.

[78] W. Nordhaus. *A Question of Balance: Weighing the options on Global Warming policy*. Yale University Press, 2008.

[79] W. D. Nordhaus and J. Boyer. Internet edition of warming the world: Economics models of global warming. `http://www.econ.yale.edu/~nordhaus/homepage/web%20table%20of%20contents%20102599.htm`.

[80] William D. Nordhaus and Joseph Boyer. *Roll the DICE Again: Economic Models of Global Warming*. `http://www.econ.yale.edu/~nordhaus/homepage/web%20pref%20102599.PDF`, 1999.

[81] Jeremy E. Oakley. Decision-theoritic sensitivity analysis for complex computer models. *Submitted to Technometrics*, 2008.

[82] Jeremy E. Oakley and Anthony O'Hagan. Probabilistic sensitivity analysis of complex models: a bayesian approach. *Journal of the Royal Statistical Society Series B*, 66(3):751–769, 2004.

[83] J. Oerlemans and C. J. van der Veen. *Ice Sheets and Climate*. D. Reidel Publishing Company, 1984.

[84] A. O'Hagan. *Bayesian Inference*. Edward Arnold, 1994.

[85] United Nations Framework Convention on Climate Change. `http://unfccc.int/kyoto_protocol/items/2830.php`.

[86] M. Oppenheimer. Global warming and the stability of the west antarctic ice sheet. *Nature*, 393, May 1998.

[87] E. Parzen. *Stochastic Processes*. Holden-Day, Inc., 1962.

[88] A. S. Peterson, D. J. Canning, T. M. Leschine, and E. L. Miles. Adapting decision making to uncertainty when addressing sea level rise response in puget sound. *Proceedings of the Coastal Zone*, 2007.

[89] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes: The art of scientific computing*. Cambridge University Press, third edition, 2007.

[90] D. Pugh. *Changing Sea Levels, Effects of Tides, Weather and Climate*. Cambridge University Press, 2004.

[91] Z. Qian and C. F. J. Wu. Bayesian heirarchical modeling for integrating low-accuracy and high accuracy experiements. *Technometrics*, 50(2):192–204, 2008.

[92] N. V. Queipo, R. T. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, and P. K. Tucker. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41:1–28, 2005.

[93] C. Radhakrishna Rao. *Linear Statistical Inference and Its Applications*. John Wiley and Sons, Inc., New York, London, Sydney, 1965.

[94] J. Rotmans and M. B. A. van Asselt. Integrated assessment: A growing child on its way to maturity. *Climatic change*, 34:327–336, 1996.

[95] J. C. Rougier. Lightweight emulators for multivariate deterministic functions. *Unpublished*, 2007.

[96] J. C. Rougier. Efficient emulators for multivariate deterministic functions. *Journal of Computational and Graphical Statistics*, 17(4):827 – 843, 2008.

[97] J. C. Rougier, S. Guillas, A. Maute, and A. D. Richmond. Emulating the thermosphere-ionosphere electrodynamics general circulation model (tie-gcm). *forthcoming*, 2008.

[98] J. C. Rougier and M. Kern. Predicting snow velocity in large chute flows under different environmental conditions. *In Submission*, 2009.

[99] J. C. Rougier, D. M. H. Sexton, J. M. Murphy, and D. Stainforth. Emulating the sensitivity of the hadsm3 climate model using ensembles from different but related experiments. *Journal of Climate*, forthcoming.

[100] J. Sacks, S. B. Schiller, and W. J. Welch. Designs for computer experiments. *Technometrics*, 31(1):41–47, 1989.

[101] J. Sacks, W. J. Welch, T. J. Mitchell, and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4:409–435, 1989.

[102] A. Saltelli, K. Chan, and E. M. Scott, editors. *Sensitivity Analysis*. John Wiley and Sons, Ltd, Chichester, 2000.

[103] T. J. Santner, B. J. Williams, and W. I. Notz. *The Design and Analysis of Computer Experiments*. Springer-Verlag New York, 2003.

[104] M. Schonlau, W. J. Welch, and D. R. Jones. Global versus local search in constrained optimization of computer models. *New Developments and applications in Experimental Design*, 34, 1998.

[105] I. C. Simpson. Numerical integration over a semi-infinite interval using the lognormal distribution. *Numerische Mathematik*, 31:71–76, 1978.

[106] T. W. Simpson, V. Toropov, V. Balabanov, and F. A. C. Viana. Design and analysis of computer experiments in multidisciplinary design optimization: A review of how far we have come - or not. *12th AIAA/ISSMO Multidisciplinary analysis and optimization conference, Victoria, British Columbia*, 10-12 September 2008, 2008.

[107] M. B. A. Van Asselt and J. Rotmans. Uncertainty in integrated assessment modelling. *Climatic Change*, 54:75–105, 2002.

[108] I. Vernon, M. Goldstein, and R. G. Bower. Galaxy formation: a bayesian uncertainty analysis. Technical report, University of Durham, 2010. Submitted to Bayesian Analysis.

[109] W. J. Welch, R. J. Buck, J. Sacks, H. P. Wynn, T. J. Mitchell, and M. D. Morris. Screening, predicting and computer experiments. *Technometrics*, 34(1):15–25, 1992.

[110] Martin Wilck. A general approximation method for solving integrals containing a lognormal weighting function. *Journal of Aerosol Science*, 32:1111–1116, 2001.

[111] B. J. Williams, T. J. Santner, and W. I. Notz. Sequential design of computer experiments to minimize integrated response functions. *Statistica Sinica*, 10:1133–1152, 2000.

Appendix A

Notation

A.1 Use of subscripts and the manipulation of arrays

Throughout this thesis we use subscripts to denote particular elements of vector, matrix or array type objects. An object with a single subscript is a vector with the subscript labelling the element, for example, a_j is the jth element of vector a. An object with a double subscript is a matrix with a particular entry labelled by the subscript, for example, A_{ij} represents the ijth element of matrix A. The meaning of objects with 3 or 4 subscripts is given on page 19, as is the convention we use for repeated subscripts in array computations. The meaning of the '$*$' operator within array computations is given on page 34.

There are instances where subscripts are used in a way that deviates from general description above. These are clearly defined in the text and also appear in the nomenclature below. One common deviation is to use subscripts that are not combinations of simple letters like ijk, but have further subscripts. For example, the subscript t_i, which appears throughout our policy making methodology, indicates the vector value of a decision or system observation (depending to which object it is added as a subscript) at a particular time in the future. This is defined on page 46 and in the nomenclature.

We define the inverse variance of an array A, $Var\,[A]^{-1}$, using the following

convention. Let $P = Var\,[A]^{-1}$, then if A is a matrix, we define P so that

$$P_{ijkl} Cov\,[A_{kl}, A_{qr}] = \begin{cases} 1 & \text{if } i = q \text{ and } j = r \\ 0 & \text{otherwise} \end{cases}$$

Similarly, if A is a three dimensional array, then we define P so that

$$P_{ijklqr} Cov\,[A_{lqr}, A_{svw}] = \begin{cases} 1 & \text{if } i = s,\ j = v \text{ and } k = w \\ 0 & \text{otherwise} \end{cases}$$

A.2 Nomenclature

$A(\theta^m, z^m)$ An expected loss as a function of all $m+1$ observations and decisions. Defined formally by (3.1) on page 54.

$A(\theta^m, z^m, H^{[m]})$ An expected loss written as a function of all $m+1$ system observations, the $m+1$ decisions, and the m observed adjusted expectations of the model emulator coefficients for the m improved models. This definition appears in chapter 5 on page 188 and applies to the decision problem where we know we can build improved versions of our current simulator in the future.

$B^m(\theta^{m-1}, z^m)$ The minimum of $A(\theta^m, z^m)$ taken over the last decision θ_{t_m}. Defined formally by (3.2) on page 55.

$B^k(\theta^{k-1}, z^k)$ The minimum of $C^{k+1}(\theta^k, z^k)$ taken over the kth policy intervention θ_{t_k} for $k = 1, \ldots, m$. Defined formally by (3.4) on page 57.

$B^m_{\lambda^m}(\theta^{m-1}, z^m)$ An upper bound on $B^m(\theta^{m-1}, z^m)$ obtained by fixing $\theta_{t_m} = \lambda_{t_m}$. This is defined by (3.17) on page 69.

$B^m_{\lambda^m}(\theta^{m-1}, z^m, H^{[m]})$ The equivalent of $B^m_{\lambda^m}(\theta^{m-1}, z^m)$ in chapter 5 for the case where we may observe the adjusted coefficients of m improved models in the future. This is defined on page 188.

$B^m_{\lambda^{k+1}}(\theta^k, z^m)$ is defined to be $A((\theta^k, \lambda^{k+1}), z^m)$, by (3.19) on page 71.

$B^m_{\lambda^{k+1}}(\theta^k, z^m, H^{[m]})$ is defined to be $A((\theta^k, \lambda^{k+1}), z^m, H^{[m]})$, by (5.36) on page 188.

$C^m(\theta^{m-1}, z^{m-1})$ The expectation of $B^m(\theta^{m-1}, z^m)$ with respect to z_{t_m} given all other decisions and observations. Defined formally by (3.3) on page 55.

$C^m_{\lambda^m}(\theta^{m-1}, z^{m-1})$ An emulator for an upper bound on $C^m(\theta^{m-1}, z^{m-1})$ conditioned on time t_m strategy λ_{t_m}. This is defined by (3.18) on page 70

$C^m_{\lambda^m}(\theta^{m-1}, z^{m-1}, H^{[m-1]})$ The equivalent of $C^m_{\lambda^m}(\theta^{m-1}, z^{m-1})$ in chapter 5 for the case where we may observe the adjusted coefficients of m improved models in the future. This is defined on page 188.

$C^k(\theta^{k-1}, z^{k-1})$ The expectation of $B^k(\theta^{k-1}, z^k)$ with respect to z_{t_k} given all previously taken decisions and observations for $k = 1, \ldots, m$. Defined formally by (3.5) on page 57.

$C^m_{\lambda^{k+1}}(\theta^k, z^{m-1})$ An emulator for an upper bound on $C^m(\theta^{m-1}, z^{m-1})$ conditioned on strategies from time t_{k+1} to time t_m of $\lambda_{t_{k+1}}, \ldots, \lambda_m$. This is defined by (3.20) on page 71.

$C^m_{\lambda^{k+1}}(\theta^k, z^{m-1}, H^{[m-1]})$ The equivalent of $C^m_{\lambda^{k+1}}(\theta^k, z^{m-1})$ in chapter 5 for the case where we may observe the adjusted coefficients of m improved models in the future. This is defined on page 188.

$C^j_{\lambda^{k+1}}(\theta^k, z^{j-1})$ An emulator for an upper bound on $C^j(\theta^{j-1}, z^{j-1})$ for $j = k+1, \ldots, m-1$, conditioned on strategies from time t_{k+1} to time t_m of $\lambda_{t_{k+1}}, \ldots, \lambda_m$. This is defined by (3.21) on page 71.

$C^j_{\lambda^{k+1}}(\theta^k, z^{j-1}, H^{[j-1]})$ The equivalent of $C^j_{\lambda^{k+1}}(\theta^k, z^{j-1})$ in chapter 5 for the case where we may observe the adjusted coefficients of m improved models in the future. This is defined on page 189.

$C^1(\theta_{t_0}, z_{t_0})$ The expected loss surface as a function of today's policy θ_{t_0} and historical system observations z_{t_0}.

$C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$ is an emulator for our upper bound on the expetced loss surface for policies made today.

e_{t_0} Is the measurement error encountered when observing real-world system data z_{t_0}, introduced on page 26.

$\tilde{E}[\cdot]$ is an operator that indicates an expectation to be obtained by emulating the argument function and evaluating the expectation on the emulator. Introduced on page 70.

$f(\cdot)$ Denotes the vector valued output for a computer simulator, first introduced on page 6.

$f^*(\cdot)$ Denotes the vector valued output of the reified simulator which is defined on page 166.

$f^0(\cdot)$ Denotes the vector valued output of our current computer model in the case where we consider the potential for improved models in chapter 5. This description is first introduced on page 168.

$f^k(\cdot)$ Dentoes the vector valued output of the improved version of our computer model that we will have access to at time t_k for $k = 1, \ldots, m$. This is defined on page 168.

$\hat{f}$ Denotes the hat run $\hat{f} = f(\hat{x})$ for a computer model that does not have any decision inputs. This is defined on page 208.

$\hat{f}(\theta)$ Denotes the hat function $\hat{f}(\theta) = f(\hat{x}, \theta)$, defined on page 216.

F Denotes a matrix whose columns are n computer model runs at locations $(x_1, \theta_1), \ldots, (x_n, \theta_n)$, first seen on page 12 and defined formally on page 19.

F^k Denotes a matrix whose columns are n runs of model $f^k(x, \theta)$ at locations $(x_1, \theta_1), \ldots, (x_n, \theta_n)$. Defined on page 173.

$g(x, \theta)$ Denotes vector of basis functions in an emulator, introduced on page 11. In chapter 5 this is generalized to be the vector of all basis functions included on any one of our emulators in a structural reification of a current computer model. This notation only applies to chapter 5 and is presented on page 169.

G Denotes the model matrix with columns $g(x_1, \theta_1), \ldots, g(x_n, \theta_n)$, defined on page 19.

H^k Is defined to be the expectation of coefficients β^k, adjusted by runs F^k. It is treated as a random quantity that may be observed in place of F^k. This is defined and explained on page 175.

$H^{[k]}$ $= H^1, \ldots, H^k$, defined on page 175.

$L(x_L, y, \theta)$ The loss model, evaluated at parameters x_L, with policy θ and complex system state y. This was defined on page 47. Throughout most of the thesis we suppress x_L as the loss model is often treated as a function of just y and θ.

m The number of intervention points for the policy support problem, defined on page 46.

t_0 when written as a subscript, indicates the subset of the labeled object that refers to historical values. For example, y_{t_0} indicates the historical state vector for the complex system y. Defined on page 26.

t_i The ith intervention point. Defined on page 46

t_f A subscript representing the labelled quantity at all future time points following a final intervention. Defined on page 47.

$u(x, \theta)$ Denotes the mean zero residual process in an emulator, introduced on page 11.

$u^k(x, \theta)$ Denotes the mean zero residual process in our emulator for the kth improved version of the simulator we are considering in chapter 5 for $k = 1, \ldots, m$. It appears first on page 171.

$u^*(x, \theta)$ Denotes the mean zero residual process in our emulator for the reified simulator. It appears first on page 171.

U Denotes the residual matrix with columns $u(x_1, \theta_1), \ldots, u(x_n, \theta_n)$, defined on page 19.

v From chapter 6 only, is part of the expression $\hat{x} = v + Wz_{t_0}$, where the values of v and W are derived from (6.1) on page 208.

W From chapter 6 only, is part of the expression $\hat{x} = v + Wz_{t_0}$, where the values of v and W are derived from (6.1) on page 208. Prior to chpater 6, W may appear as a locally defined variable and it's meaning is to be taken in the context of the passage in which it appears.

x Denotes model inputs for a computer simulator unless otherwise stated, first seen on page 7. In chapter 5, when considering the potential for improved models, we extend x to be the vector of all of the model inputs that we can possibly conceive of including on any version of the simulator. This is described in detail on page 168.

x^* Denotes the best input for a computer model, defined on page 8. The meaning of x^* is changed in chapter 5 where it is defined to be the best input for the reified simulator only. This change in definition, occurring on page 166, only applies to the situation where we have improved models in chapter 5.

x_L The set of non-decision or system parameters for a loss model, defined on page 47.

$\hat{x}$ The expectation of x^* adjusted by observations of the complex system z_{t_0}. This is defined by (6.1) on page 208.

X Denotes the space of possible model inputs, first seen on page 8.

y Denotes the state vector of the complex system and is first seen on page 8.

z_{t_0} Represents observations of the complex system y, introduced on page 25.

z_{t_i} The system observation made at the ith intevention point, defined on page 47.

z^k The collection of observations of the complex system made, including historical observations and up to the kth intervention point. Defined on page 53.

β Denotes the matrix of coefficients to the regression surface in an emulator, introduced on page 11.

β^0 Denotes the matrix of coefficients to the regression surface of our current emulator in chapter 5 when we consider the potential for observation of runs on improved models. This is introduced on page 169.

β^k Denotes the matrix of coefficients to the regression surface of our emulator for the kth improved version of the simulator we are considering in chapter 5 for $k = 1, \ldots, m$. It is extended to be compatible with the vector of basis functions, $g(x, \theta)$, common to the emulators of all of the improved models. This is defined and described on page 171.

β^* Denotes the matrix of coefficients to the regression surface of our emulator for the reified model. It is extended to be compatible with the vector of basis functions, $g(x, \theta)$, common to the emulators of all of the models under consideration. The first appears on page 171.

Δ $= F - E[F]$ introduced on page 20.

$\eta(\theta)$ Represents the discrepancy of our current model from the complex system y. It is defined formally on page 8 in chapter 2.

$\eta^*(\theta)$ Represents the discrepancy of the reified model and is defined on page 166 in chapter 5.

θ Denotes the vector of decision parameters as inputs to a computer simulator, first introduced on page 7.

θ_{t_0} The policy that is to be made today, defined on page 46.

θ_{t_i} The decision to be made at the ith intervention point, defined on page 47.

θ^k The collection of decisions taken from now, up to and including intervention k. Defined on page 53.

Θ The design matrix for the decision inputs, defined on page 34.

Θ_i The ith column of the design matrix for the decision inputs. This is a deviation from the usual convention on subscripts and is introduced on page 34.

$\lambda_{t_i}(\theta^{i-1}, z^i)$ The time t_i strategy we fix during Sequential Emulation. It is written as a function of previous decisions and observations of the system and represents the emulator driven time t_i strategy. Defined on page 69, and again in a Sequential Emulation setting on page 69 and 71 for $i = m$ and $i < m$ respectively. After its introduction, this latter definition applies for the rest of the thesis. We often use λ_{t_i} as shorthand for this function. Note that in chapter 5, this shorthand notation implies $\lambda_{t_i}(\theta^{i-1}, z^i, H^{[i]})$, which is defined below.

λ^k is defined as the collection of all interventions from the kth intervention until the final one. That is $\lambda^k = \lambda_{t_k}, \ldots, \lambda_{t_m}$ which is defined on page 71.

$\lambda_{t_i}(\theta^{i-1}, z^i, H^{[i]})$ The time t_i strategy we fix during Sequential Emulation for the case where we know we may observe runs on improved models in the future. It is written as a function of previous decisions, observations of the system, and observed adjusted coefficients on improved models. It represents the emulator driven time t_i strategy in chapter 5 and is defined by (5.33) on page 188 and (5.39) on page 189 for $i = m$ and $i < m$ respectively. We use λ_{t_i} as shorthand for this function within chapter 5. Aside from within chapter 5, the notation λ_{t_i} refers to the time t_i strategy for the decision problem where we only consider our current simulator.

Σ Denotes the variance matrix of $u(x, \theta)$ at one value of x and θ, first seen on page 11.

Ψ Denotes the vector of correlation parameters within the correlation function for a emulator Gaussian process, described on page 11.

Ω The design matrix for the model inputs, defined on page 34.

Ω_i The ith column of the design matrix for the model inputs. This is a deviation from the usual convention on subscripts and is introduced on page 34.

ω^k This is defined to be $\beta^k - H^k$ and to be independent of H^k. This is explained on page 175. The variance of this quantity turns out to be a decision variable in the Sequential Emulation algorithm of chapter 5.

Appendix B

Integrating out the best input of C-GOLDSTEIN analytically

In section 4.5.1, we demonstrated the separability of our emulator for C-GOLDSTEIN and stated that we could solve the required $x-$dependent integrals,

$$\int_{-1}^{1} h(x^*)p(x^*)dx^*, \qquad \int_{-1}^{1} h(x^*)h(x^*)^T p(x^*)dx^*,$$
$$\int_{-1}^{1} r^x(|x^* - \Omega|)p(x^*)dx^*, \qquad \int_{-1}^{1} r^x(|x^* - \Omega|)r^x(|x^* - \Omega|)^T p(x^*)dx^*,$$
$$\int_{-1}^{1} h(x^*)r^x(|x^* - \Omega|)p(x^*)dx^*,$$

where

$$h(x) = (1, x_1, x_2, x_3, 1, 1, 1, 1)^T,$$

and

$$r^x(|x - x'|) = \exp\{(x - x')\Lambda_x(x - x')^T\},$$

analytically. We illustrate these calculations here.

We have $p(x^*)$ such that

$$x_i^* \sim U(-1, 1)$$

for $i = 1, 2, 3$, and the pdf of our distribution for x^* is $\frac{1}{8}$. We begin be computing $\int_{-1}^{1} h(x^*)p(x^*)dx^*$. In order to do this we need to integrate 1 and x_i for $i = 1, 2, 3$,

with respect to our pdf for x^*. Firstly,

$$\int p(x^*)dx^* = \int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1}\frac{1}{8}dx_1^*dx_2^*dx_3^*$$
$$= 1.$$

For $i = 1, 2, 3$ we have

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1}\frac{1}{8}x_i^*dx_1^*dx_2^*dx_3^* = 0,$$

therefore

$$\int_{-1}^{1} h(x^*)p(x^*)dx^* = (1, 0, 0, 0, 1, 1, 1, 1)^T.$$

Moving on to the second of the five integrals we have to compute, we can write down

$$h(x)h(x)^T = \begin{pmatrix} 1 & x_1 & x_2 & x_3 & 1 & 1 & 1 & 1 \\ x_1 & x_1^2 & x_1x_2 & x_1x_3 & x_1 & x_1 & x_1 & x_1 \\ x_2 & x_2x_1 & x_2^2 & x_2x_3 & x_2 & x_2 & x_2 & x_2 \\ x_3 & x_3x_1 & x_3x_2 & x_3^2 & x_3 & x_3 & x_3 & x_3 \\ 1 & x_1 & x_2 & x_3 & 1 & 1 & 1 & 1 \\ 1 & x_1 & x_2 & x_3 & 1 & 1 & 1 & 1 \\ 1 & x_1 & x_2 & x_3 & 1 & 1 & 1 & 1 \\ 1 & x_1 & x_2 & x_3 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

In order to evaluate the required integral then, in addition to the calculations that we have already performed, we integrate terms such as x_i^{*2} and $x_i^*x_j^*$ for $i, j \in \{1, 2, 3\}$ and $i \neq j$. Performing these calculations we obtain

$$\int_{-1}^{1} h(x^*)h(x^*)^T p(x^*)dx^* = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix}.$$

The integral $\int_{-1}^{1} r^x(|x^* - \Omega|)p(x^*)dx^*$ is a vector whose length is equal to the number of columns in Ω. The kth element of this vector is

$$\left[\int r^x(|x^* - \Omega|)p(x^*)dx^*\right]_k = \int r^x(|x^* - \Omega_k|)p(x^*)dx^*.$$

In order to compute this we must evaluate

$$\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \frac{1}{8} e^{-(x^*-\Omega_k)\Lambda_x(x^*-\Omega_k)^T} dx_1^* dx_2^* dx_3^*.$$

Let

$$\Lambda_x = \begin{pmatrix} \Lambda_1 & 0 & 0 \\ 0 & \Lambda_2 & 0 \\ 0 & 0 & \Lambda_3 \end{pmatrix},$$

then the required integral becomes

$$\frac{1}{8}\int_{-1}^{1} e^{-\Lambda_1(x_1^*-\Omega_{1k})^2} dx_1^* \int_{-1}^{1} e^{-\Lambda_2(x_2^*-\Omega_{2k})^2} dx_2^* \int_{-1}^{1} e^{-\Lambda_3(x_3^*-\Omega_{3k})^2} dx_3^*,$$

which is the product of 3 integrals. Now, for $i = 1, 2, 3$ let

$$\Psi_i = \frac{1}{\sqrt{2\Lambda_i}}.$$

We can then write the ith integral in the product above as

$$\begin{aligned}
\int_{-1}^{1} e^{\frac{-(x_i^*-\Omega_{ik})^2}{2\Psi_i^2}} dx_i^* &= \int_{-\infty}^{1} e^{\frac{-(x_i^*-\Omega_{ik})^2}{2\Psi_i^2}} dx_i^* - \int_{-\infty}^{-1} e^{\frac{-(x_i^*-\Omega_{ik})^2}{2\Psi_i^2}} dx_i^* \\
&= \sqrt{2\pi}\Psi_i \left(\frac{1}{\sqrt{2\pi}\Psi_i} \int_{-\infty}^{1} e^{\frac{-(x_i^*-\Omega_{ik})^2}{2\Psi_i^2}} dx_i^* - \frac{1}{\sqrt{2\pi}\Psi_i} \int_{-\infty}^{-1} e^{\frac{-(x_i^*-\Omega_{ik})^2}{2\Psi_i^2}} dx_i^* \right) \\
&= \sqrt{2\pi}\Psi_i \left(\Phi\left(\frac{1-\Omega_{ik}}{\Psi_i}\right) - \Phi\left(\frac{-1-\Omega_{ik}}{\Psi_i}\right)\right),
\end{aligned}$$

where $\Phi(\cdot)$ is the cdf of the standard normal distribution. Therefore

$$\left[\int r^x(|x^* - \Omega|)p(x^*)dx^*\right]_k = \frac{\pi^{\frac{3}{2}}}{2\sqrt{2}} \prod_{i=1}^{3} \Psi_i \left(\Phi\left(\frac{1-\Omega_{ik}}{\Psi_i}\right) - \Phi\left(\frac{-1-\Omega_{ik}}{\Psi_i}\right)\right).$$

The integral $\int_{-1}^{1} r^x(|x^* - \Omega|)r^x(|x^* - \Omega|)^T p(x^*)dx^*$ is a square matrix whose size in each dimension is the same as the number of columns in Ω. We can write the lkth element of this as

$$\begin{aligned}
&\int_{-1}^{1}\int_{-1}^{1}\int_{-1}^{1} \frac{1}{8} e^{-(x-\Omega_l)\Lambda_x(x-\Omega_l)^T} e^{-(x-\Omega_k)\Lambda_x(x-\Omega_k)^T} dx_1^* dx_2^* dx_3^* \\
&= \frac{1}{8}\prod_{i=1}^{3}\int_{-1}^{1} e^{-\Lambda_i((x_i^*-\Omega_{il})^2+(x_i^*-\Omega_{ik})^2)} dx_i^*.
\end{aligned}$$

We solve this by first completeing the square on $(x_i^* - \Omega_{il})^2 + (x_i^* - \Omega_{ik})^2$, for $i = 1, 2, 3$.

$$\begin{aligned}(x_i^* - \Omega_{il})^2 + (x_i^* - \Omega_{ik})^2 &= x_i^* - 2x_i^*\Omega_{il} + \Omega_{il}^2 + x_i^* - 2x_i^*\Omega_{ik} + \Omega_{ik}^2 \\ &= 2(x_i^* - \frac{(\Omega_{il} + \Omega_{ik})}{2})^2 + \frac{1}{2}(\Omega_{il} - \Omega_{ik})^2.\end{aligned}$$

We have written our required integral as the product of three integrals. Having completed the square on the exponent of each integrand, we can express the ith integral as

$$e^{-\frac{\Lambda_i}{2}(\Omega_{il}-\Omega_{ik})^2} \int_{-1}^{1} e^{-2\Lambda_i(x_i^* - (\frac{\Omega_{il}+\Omega_{ik}}{2}))^2} dx_i^* = e^{-\frac{\Lambda_i}{2}(\Omega_{il}-\Omega_{ik})^2} \int_{-1}^{1} e^{\frac{-(x_i^* - (\frac{\Omega_{il}+\Omega_{ik}}{2}))^2}{2\tau_i^2}} dx_i^*$$

which can be written as

$$e^{-\frac{\Lambda_i}{2}(\Omega_{il}-\Omega_{ik})^2} \sqrt{2\pi}\tau_i \left(\Phi\left(\frac{1 - (\frac{\Omega_{il}+\Omega_{ik}}{2})}{\tau_i} \right) - \Phi\left(\frac{-1 - (\frac{\Omega_{il}+\Omega_{ik}}{2})}{\tau_i} \right) \right),$$

where $\tau_i = \sqrt{\frac{1}{4\Lambda_i}}$ for $i = 1, 2, 3$. Therefore, letting

$$W_{lk} = \int r^x(|x^* - \Omega_l|) r^x(|x^* - \Omega_k|)^T p(x^*) dx^*,$$

we have

$$W_{lk} = \frac{\pi^{\frac{3}{2}}}{2\sqrt{2}} \prod_{i=1}^{3} \tau_i e^{-\frac{\Lambda_i}{2}(\Omega_{il}-\Omega_{ik})^2} \left(\Phi\left(\frac{1 - (\frac{\Omega_{il}+\Omega_{ik}}{2})}{\tau_i} \right) - \Phi\left(\frac{-1 - (\frac{\Omega_{il}+\Omega_{ik}}{2})}{\tau_i} \right) \right).$$

The final integral we have to solve is $\int_{-1}^{1} h(x^*) r^x(|x^* - \Omega|) p(x^*) dx^*$, which is a matrix with 8 rows and the same number of columns as Ω. We construct this matrix one row at a time. The first row and the last four correspond to the vector $\int_{-1}^{1} r^x(|x^* - \Omega|) p(x^*) dx^*$, whose value was derived above. The remaining three rows are $\int_{-1}^{1} x_i^* r^x(|x^* - \Omega|) p(x^*) dx^*$ for $i = 1, 2, 3$. The kth element of this vector when $i = 2$, for example, is

$$\frac{1}{8} \int_{-1}^{1} e^{-\Lambda_1(x_1^* - \Omega_{1k})^2} dx_1^* \int_{-1}^{1} x_2^* e^{-\Lambda_2(x_2^* - \Omega_{2k})^2} dx_2^* \int_{-1}^{1} e^{-\Lambda_3(x_3^* - \Omega_{3k})^2} dx_3^*.$$

In order to complete our integrations then, we must be able to solve the integral $\int_{-1}^{1} x_i^* e^{-\Lambda_i(x_i^* - \Omega_{ik})^2} dx_i^*$, for $i = 1, 2, 3$.

Writing the integrand as

$$x_i^* e^{-\Lambda_i x_i^{*2}} e^{2\Lambda_i \Omega_{ik} x_i^*} e^{-\Lambda_i \Omega_{ik}^2},$$

we can integrate by parts to obtain

$$\int_{-1}^{1} x_i^* e^{-\Lambda_i (x_i^* - \Omega_{ik})^2} dx_i^* = \left[-\frac{1}{2\Lambda_i} e^{-\Lambda_i (x_i^* - \Omega_{ik})^2} \right]_{-1}^{1} + \Omega_{ik} \int_{-1}^{1} e^{-\Lambda_i (x_i^* - \Omega_{ik})^2} dx_i^*,$$

which we write as

$$\Gamma_{ik} = \frac{e^{-\Lambda_i (-1-\Omega_{ik})^2} - e^{-\Lambda_i (1-\Omega_{ik})^2}}{2\Lambda_i} + \Omega_{ik} \sqrt{2\pi} \Psi_i \left(\Phi\left(\frac{1-\Omega_{ik}}{\Psi_i} \right) - \Phi\left(\frac{-1-\Omega_{ik}}{\Psi_i} \right) \right).$$

Therefore, the row of $\int_{-1}^{1} h(x^*) r^x(|x^* - \Omega|) p(x^*) dx^*$ corresponding to $h(x^*)_i = x_2^*$ is

$$\frac{\pi}{4} \Psi_1 \Psi_3 \left(\Phi\left(\frac{1-\Omega_{1k}}{\Psi_1} \right) - \Phi\left(\frac{-1-\Omega_{1k}}{\Psi_1} \right) \right) \left(\Phi\left(\frac{1-\Omega_{3k}}{\Psi_3} \right) - \Phi\left(\frac{-1-\Omega_{3k}}{\Psi_3} \right) \right) \Gamma_{2k}.$$

The rows corresponding to $h(x^*)_i = x_1^*$ and $h(x^*)_i = x_3^*$ are obtained similarly.

Appendix C

The DICE model

We present here the R code for the decoupled DICE model used in the examples of Chapter 4 and Chapter 5. We begin by introducing the function EmissionSeries(), which was used to generate designs for C-GOLDSTEIN as well as being a required component of the DICE model. The purpose of the function is to take values of our decisions θ_{t_0} (RateNow) and θ_t (RateInt), and use the BAU emissions curve we described in section 4.2 to calculate an emissions curve corresponding to our decision. This curve is then used as an input to C-GOLDSTEIN and is used within the DICE function to calculate the cost of abatement. The input called IntTime, which we had once considered might be used as another decision variable, represents the time t_1 at which we observe climate and make our intervention. In all of the examples we have looked at the value of the variable was 4, corresponding to $t_1 = 2035$.

```
EmissionSeries <- function(IntTime,RateNow,RateInt){
  load("10yearBAU.RData")
  BAU <- TenBAU
  time <- 0 #Year 1995
  Emission <- rep(0,length(BAU$Total))
  Emission[1] <- RateNow*BAU$Total[1]
  time <- 1
  while(time<IntTime){
    Emission[time+1]=RateNow*(BAU$Total[time+1]-(BAU$Total[time]-Emission[time]))
    time = time +1
  }
  for(time in IntTime:(length(Emission)-1)){
```

```
    Emission[time+1]=RateInt*(BAU$Total[time+1]-(BAU$Total[time]-Emission[time]))
  }
  return(Emission)
}
```

Here is the decoupled DICE model coded in R. Annotations follow the # symbol. Most of the inputs to the DICE model are parameters controlling the evolution of various parts of the model. $L0$ and *delpop* for example control the evolution of the global population. The default values we have given for each of these variables was obtained via the excel version of the full DICE-99 model, freely available from [79]. We describe the functionality of the inputs that we have added to our own version of DICE only.

```
MODEL INPUTS:
Yh: Global temperature in 1995
Yt: Global temperature in 2035
Yp: Global temperature in 2095
TempSeries: A time series of global temperatures from 1995 to 2095.
            If NULL a series is constructed by interpolating Yh, Yt and Yp.
IntTime: See the function EmissionSeries() above.
RateNow: See the function EmissionSeries() above.
RateInt: See the function EmissionSeries() above.
Consumption: Logical variable. If TRUE, in addition to a utility,
             we return a time series of global consumption.

DICE <- function(Yh,Yt,Yp,TempSeries=NULL,IntTime,RateNow,RateInt,rho0=0.03,
        gp=0.00257,gpop0=0.157,decades=10,L0=5632.7,delpop=0.222, futureRate=0.1,
        sRates=c(25.3,24.02,23.27,22.81,22.52,22.35,22.25,22.21,22.20,22.23,22.29),
        theta1 = -0.004500,theta2=0.003500,b10=0.03,gb0=-0.08,delb=0.005,b2=2.150,
        A0=0.017,gA0=0.038,delA=0,K0=47,delk=0.1,gamma=0.3,Consumption=FALSE){

 if(is.null(TempSeries)){
   TempSeries <- rep(0,11)
   TimeIndex <- 0
   mt <- (Yt - Yh)/4
   mp <- (Yp - Yt)/6
   while(TimeIndex < 11){
```

```
    if(TimeIndex < 5){
      TempSeries[TimeIndex + 1] <- Yh + (mt*TimeIndex)
    }
    else{
      TempSeries[TimeIndex + 1] <-  Yt + (mp * (TimeIndex-4))
    }
      TimeIndex <- TimeIndex + 1
   }
}
Emitted <- EmissionSeries(IntTime=IntTime,RateNow=RateNow,RateInt)
load("10yearBAU.RData")
BAU <- TenBAU$Total

model <- function(time,lastRate,lastStock,lastInvest,lastAbateCoeff){

 #Time t discount rate
 rhot <- rho0*exp(-10*gp*time)
 Rt <- lastRate/((1+rhot)^10)
 if(time==0){
   Rt=lastRate
 }
 #Time t population
 Lt <- L0*exp((gpop0/delpop)*(1-exp(-delpop*time)))

 #Time t Damage
 Temp <- TempSeries[time+1] #time starts from 0.
 Dt <- (theta1*Temp)+(theta2*(Temp^2))

 #Time t damage factor on output
 Omegat <- 1/(1+Dt)

 #Time t Total factor productivity
 if(delA==0){
   At <- A0*exp(gA0*time)
 }
 else{
   At <- A0*exp((gA0/delA)*(1-exp(-delA*time)))
```

```
  }

  #Time t Capital stocks
  Kt <- (lastStock*((1-delk)^10)) + (10*lastInvest) #check units of K0
  if(time==0){
    Kt = lastStock
  }

  #Time t emission control rate
  ut <- 1 - (Emitted[1 + time]/BAU[1 + time])

  #Time t abatement cost
  gbt <- gb0*exp(-delb*time)
  b1t <- lastAbateCoeff/(1+gbt)
  if(time==0){
    b1t = b10
  }
  abate <- 1 - (b1t*(ut)^b2)

  #Time t Global Economic Output
  Qt <- Omegat*abate*At*(Kt^gamma)*(Lt^(1-gamma))

  #Time t Investment
  It <- Qt*sRates[time+1]/100

  #Time t consumption
  Ct <- Qt-It

  #Time t consumption per capita
  ct <- Ct/Lt

  #Time t utility
  utilt <- Lt*log(ct)

  return(list(Ct=Ct,Rt=Rt,tUtility=utilt,NewInvest=It,NewStock=Kt,NewAbate=b1t))
}
```

```
#We're going to start the model at time 0, so here we create the time -1 values
#lastRate=lR,lastStock=lS,lastAbateCoeff=lA,lastInvest=lI
   lR <- 1
   lI <- 0
   #LastStock and lastAbateCoeff are already model parameters
   lS <- K0
   lA <- b10
   Time <- 0
   W <- 0
   utilitySeries <- c()
   if(Consumption)
     Con <- c()
   #Here we run DICE from 1995 to 2095 in decade sized steps
   while(Time<=decades){
    M <- model(time=Time,lastRate=lR,lastStock=lS,lastInvest=lI,lastAbateCoeff=lA)
    utilitySeries <- c(utilitySeries,M$tUtility)
    if(Consumption)
      Con <- c(Con,M$Ct)
    W <- W+(M$tUtility * M$Rt)
    lR <- M$Rt
    lS <- M$NewStock
    lI <- M$NewInvest
    lA <- M$NewAbate
    Time <- Time+1
   }
   fR <- futureRate
   if(Consumption)
     return(list(loss=-(W+(lR*utilitySeries[decades+1]*(1+fR)/(fR))),Con=Con))
   else
     return(W+(lR*utilitySeries[decades+1]*(1+fR)/(fR)))
}
```

The DICE model was altered slightly for the example in chapter 5. We included a build cost function and a run cost function. The additional inputs were *VRho-Prime*= $Var\left[\rho'\right]$ as defined on page 192, *Vw*= $Var\left[\omega^1\right]$, and two values *BuildCost* and *RunCost*. The following lines were inserted into the code before the *model* function.

```
#Compute alpha, the percentage of Vrho unresolved
alpha <- Vw/VRhoPrime
#Run cost will tends to infinity as alpha tends to 0.
#Bound alpha from below in order to bound the run cost.
if(alpha < 0.00005)
  alpha <- 0.00005
```

The investment section of the model was then augmented in the following way:

```
#Time t Investment
It <- Qt*savingsRates[time+1]/100
if(time==IntTime){
  if(alpha < 1)
    It <- It + BuildCost - RunCost*log(alpha)
}
```

Appendix D

Selected annotated code for forecasting and Sequential Emulation

In this appendix we present some selected R-code developed for the implementation of our methodology. Whilst much of the code is written in full generality, some pieces were designed solely to work for the examples we have presented in this thesis.

D.1 Decision-dependent forecasting

In this section we present the code used to obtain forecasts in the case where the emulator is fully separable. This code is in six parts. The first part establishes the prior emulator and enables us to compute the quantities that are required for adjusting the emulator, such as $Var[F]$. The second part contains code used to perform the Bayes Linear adjustment of our emulator by the model runs F. The third uses separability to compute the integrals necessary for forecasting. In the fourth part we use the integrals we have calculated in order to derive beliefs about the complex system. The fifth part takes our beliefs about the system and any observations and computes the decision-dependent forecast. In the sixth part we include wrapper functions drawing all of the forecasting code together in order to compute a forecast for a particular value of θ.

D.1.1 The prior emulator

We require known design matrices, a known form of $g(x, \theta)$, $E[\beta]$,$Var[\beta]$, and all of the properties of the autocovariance of the residual.

```
#g is a known vector of monomials in x and theta. This is specified by the user.
#Here we use the vector of monomials defined in chapter $4$.
g <- function(x,theta){
  c(1,x[1],x[2],x[3],theta[1],theta[1]^2,theta[2]^2,theta[1]*theta[2])
}

#G will be the model matrix for the chosen design in x and theta
G <- function(X,Theta){
#X is mxn
#Theta is pxn
#n is the number of points taken from each space,
#m is the length of x, p is the length of theta.
  if(!(length(X[1,])==length(Theta[1,]))){
    stop("Incompatible design matrices")}
     n <- length(X[1,])
     sapply(1:n,function(i) g(X[,i],Theta[,i]))
}
```

We now compute the prior variance of the model runs, $Var[F]$. This calculation comes in two parts. We have the variance of the regression surface, which is calculated using the function *VBG*, and the variance matrix for the residuals $Var[U]$. In order to calculate the latter we use equation 4.7. A feature of this equation, discussed in the main text of chapter 4, is the separability of the covariance function in x and θ. We use separate correlation functions for x and θ, coded using the functions *ModcorrFun* and *DeccorrFun* respectively. The two correlation functions we used were the Gaussian correlation functions, and as input they both take two points in the input space and a vector of correlation parameters. The function *VarU* calculates the prior variance of the residuals for the model runs, and the function *VarF* calls this function in order to calculate the prior variance of the model runs.

```
VBG <- function(G,VarBeta){
  library(tensor)
```

```
  tmp <- tensor(VarBeta,G,4,1)
  vbg <- tensor(tmp,G,2,1)
  aperm(vbg,c(1,4,2,3))
}

ModcorrFun <- function(x1,x2,CLx){
   #CLx is the vector of correlation parameters
  n <- length(CLx)
  if(!(length(x1)==length(x2)))
    stop("Incompatible input vectors")
  if(!(length(x1)==n))
     stop("Wrong number of correlation lengths provided")
  diff <- abs(x1-x2)
  C <- diag(CLx,nrow=n,ncol=n)
  exponent <- t(diff)%*%C%*%diff
  exp(-exponent)
}

DeccorrFun <- function(theta1,theta2,CLdec){
  #CLdec is vector of correlation parameters
  n <- length(CLdec)
  if(!(length(theta1)==length(theta2)))
    stop("Incompatible input vectors")
  if(!(length(theta1)==n))
    stop("Wrong number of correlation lengths provided")
  diff <- abs(theta1-theta2)
  C <- diag(CLdec,nrow=n,ncol=n)
  exponent <- t(diff)%*%C%*%diff
  exp(-exponent)
  }

VarU <- function(X,Theta,Rx,Rdec,CLx,CLdec,SIGu,V,DecisionVector){
 #The input DecisionVector controls how correlation on each output is
 #calculated by describing which outputs depend on which decisions.
 #The vector is the same length as the number of outputs.
 #It's last element should be the total number of decisions.
 #When 0 the corresponding model output doesn't depend on decisions.
```

```
#When 1 the corresponding output has dependence on decision 1.
#When 2 the corresponding output has dependence on decision 1 AND 2
#And so on up to the number of decisions
 stopifnot(is.function(Rx))
 stopifnot(is.function(Rdec))
 m <- length(SIGu)
 n <- length(X[1,])
 if(!(length(Theta[1,])==n))
   stop("Incompatible Design Matrices")
 numDecs <- length(Theta[,1])
 if(!(length(DecisionVector)==m))
   stop("DecisionVector must be the same length as the output")
 if((DecisionVector[m]>numDecs))
   stop("Decision Vector specifies more decisions than required")
 if(DecisionVector[m]<numDecs)
   stop("We have decisions we are not using")
 VU <- array(0,c(m,n,m,n))
 for(i in 1:m){
   for(j in 1:n){
     for(k in 1:m){
       for(l in 1:n){
         minOutput <- min(i,k)
         if(DecisionVector[minOutput]<1)
           VU[i,j,k,l] <- V[i,k]*SIGu[i]*SIGu[k]*Rx(x1=X[,j],x2=X[,l],CLx=CLx)
         else
           VU[i,j,k,l] <- V[i,k]*SIGu[i]*SIGu[k]*Rx(x1=X[,j],x2=X[,l],CLx=CLx)
                          *Rdec(theta1=Theta[1:DecisionVector[minOutput],j],
                          theta2=Theta[1:DecisionVector[minOutput],l],
                          CLdec=CLdec[1:DecisionVector[minOutput]])
         }
       }
     }
   }
   return(VU)
 }

VarF <- function(X,Theta,VarBeta,Rx,Rdec,CLx,CLdec,SIGu,V,DecisionVector){
```

```
  ourG <- G(X,Theta)
  vbg <- VBG(ourG,VarBeta)
  vu <- VarU(X,Theta,Rx,Rdec,CLx,CLdec,SIGu,V,DecisionVector)
  vbg + vu
}
```

D.1.2 Adjusting the prior emulator

In order to compute the Bayes Linear adjustment to the emulator we must invert the array $Var[F]$, and perform a number of array manipulations. In order to invert arrays we convert them into appropriate matrices and use one of the standard methods of matrix inversion, before converting the answer back into an array. A particularly robust form of matrix inversion in R involves computing the Cholesky factors and then performing the matrix multiplication with these. So for example, if we require the product $xV^{-1}y$, then we find upper triangular Q such that $V = Q^TQ$ and we compute $xV^{-1}y = A^TB$, where $A^T = xQ^{-1}$ and $B = Q^{-T}y$. Most of the array manipulations that involve a matrix inversion throughout this code attempt to handle the inverse in this way. Often, some of the eigenvalues are very small and the Cholesky factors of the matrix we wish to invert cannot be found. If the attempt to perform this particular type of robust matrix inversion fails, we use a generalized inverse in R called *ginv()* to perform the inversion.

We calculate the quantity $F - E[F]$ via the function *Fdiff*. The function *HalfUpdate* computes $Q^{-T}(F - E[F])$, where $Var[F] = Q^TQ$.

```
Fdiff <- function(EBeta,X,Theta,Data){
  ourG <- G(X,Theta)
  library(tensor)
  EF <- tensor(EBeta,ourG,2,1)
  if(!(length(Data[1,])==length(EF[1,])))
    stop("Data and its expectation are incompatible arrays")
  if(!(length(Data[,1])==length(EF[,1])))
    stop("Data and its expectation are incompatible arrays")
  Data - EF
}
```

```
tCholFactors <- function(VarData,tChol){
  Vdim = dim(VarData)
  dim(VarData) <- c(Vdim[1]*Vdim[2],Vdim[3]*Vdim[4])
  Q <- chol(VarData)
  dim(Q) <- c(Vdim[1:4])
  return(Q)
}

HalfUpdate <- function(Q,FminusEF){
  Qdim <- dim(Q)
  Diffdim <- dim(FminusEF)
  dim(Q) <- c(Qdim[1]*Qdim[2],Qdim[3]*Qdim[4])
  dim(FminusEF) <-  Diffdim[1]*Diffdim[2]
  ans <- backsolve(Q,FminusEF,transpose=TRUE)
  dim(ans) <- c(Qdim[1],Qdim[2],1,1)
  return(ans)
}
```

The functions *AdjEBetas* and *AdjVarBetas* compute the adjusted expectation and the adjusted variance of β.

```
AdjEBetas <- function(EBeta,Q,X,Theta,HalfUp,VarBeta,InvVDat,FminusEF){
  #check that either Q and HalfUp or InvVDat is not NULL
  if((is.null(Q) || is.null(HalfUp)) && is.null(InvVDat))
    stop("Q and InvVDat NULL. Require information about the Data Variance")
  if(!(is.null(Q)==is.null(HalfUp)))
    stop("Q or HalfUp are non-trivial but not both")
  ourG <- G(X,Theta)
  library(tensor)
  varBG <- tensor(VarBeta,ourG,4,1)
  vbgdim <- dim(varBG)
  if(is.null(InvVDat)){
    dim(varBG) <- c(vbgdim[1]*vbgdim[2],vbgdim[3]*vbgdim[4])
    Qdim <- dim(Q)
    dim(Q) <- c(Qdim[1]*Qdim[2],Qdim[3]*Qdim[4])
    y <- backsolve(Q,t(varBG),transpose=TRUE)
    x <- HalfUp
    xdim <- dim(x)
```

```
    dim(x) <- c(xdim[1]*xdim[2],xdim[3]*xdim[4])
    adjustment <- crossprod(y,x)
    dim(adjustment) <- c(vbgdim[1],vbgdim[2])
    return(EBeta+adjustment)
  }
  else{
    RHSofAdjustment <- tensor(InvVDat,FminusEF,c(3,4),c(1,2))
    stopifnot(all(dim(RHSofAdjustment)==c(vbgdim[3],vbgdim[4])))
    Adjustment <- tensor(varBG,RHSofAdjustment,c(3,4),c(1,2))
    stopifnot(all(dim(Adjustment)==c(vbgdim[1],vbgdim[2])))
    return(EBeta+Adjustment)
  }
}

AdjVarBetas <- function(VarBeta,Q,X,Theta,InvVDat){
  if(is.null(Q) && is.null(InvVDat))
    stop("Q and InvVDat NULL at the same time")
  ourG <- G(X,Theta)
  library(tensor)
  A <- tensor(VarBeta,ourG,4,1)
  B <- tensor(VarBeta,ourG,2,1)
  B <- aperm(B,c(1,4,2,3))
  if(is.null(InvVDat)){
    Qdim <- dim(Q)
    dim(Q) <- c(Qdim[1]*Qdim[2],Qdim[3]*Qdim[4])
    #looking for AQ^-1Q^-TB
    Adim <- dim(A)
    dim(A) <- c(Adim[1]*Adim[2],Adim[3]*Adim[4])
    Bdim <- dim(B)
    dim(B) <- c(Bdim[1]*Bdim[2],Bdim[3]*Bdim[4])
    y <- backsolve(Q,t(A),transpose=TRUE)
    x <- backsolve(Q,B,transpose=TRUE)
    ans <- crossprod(y,x)
    dim(ans) <- c(Adim[1],Adim[2],Bdim[3],Bdim[4])
    ans <- VarBeta-ans
    return(ans)
  }
```

```
  else{
    #looking for AV(D)^{-1}B
    RHSofAdjustment <- tensor(InvVDat,B,c(3,4),c(1,2))
    stopifnot(all(dim(RHSofAdjustment)==
       c(dim(InvVDat)[1],dim(InvVDat)[2],dim(B)[3],dim(B)[4])))
    Adjustment <- tensor(A,RHSofAdjustment,c(3,4),c(1,2))
    stopifnot(all(dim(Adjustment)==
       c(dim(A)[1],dim(A)[2],dim(B)[3],dim(B)[4])))
    return(VarBeta - Adjustment)
  }
}
```

To complete this section we provide a function, $CovuxU$, that computes $Cov\left[u(x,\theta),U\right]$, for any x and θ. This is required in order to compute the full Bayes Linear adjustment of $f(x,\theta)$ by F.

```
CovuxU <- function(x,theta,X,Theta,V,SIGu,Rx,Rdec,CLx,CLdec,DecisionVector){
  stopifnot(is.function(Rx))
  stopifnot(is.function(Rdec))
  m <- length(SIGu)
  n <- length(X[1,])
    if(!(length(Theta[1,])==n))
      stop("Incompatible Design Matrices")
    numDecs <- length(Theta[,1])
    if(!(length(DecisionVector)==m))
      stop("The Decision Vector is not the same length as number of outputs")
    if((DecisionVector[m]>numDecs))
      stop("Decision Vector specifies more decisions than we have in Theta")
    if(DecisionVector[m]<numDecs)
      stop("We have decisions we are not using")
    CUx <- array(0,dim=c(m,m,n))
    for(i in 1:m){
      for(j in 1:m){
        for(k in 1:n){
          minOutput <- min(i,j)
          if(DecisionVector[minOutput]<1)
            CUx[i,j,k] <- V[i,j]*SIGu[i]*SIGu[j]*Rx(x1=x,x2=X[,k],CLx=CLx)
          else
```

```
            CUx[i,j,k] = V[i,j]*SIGu[i]*SIGu[j]*Rx(x1=x,x2=X[,k],CLx=CLx)*
                         Rdec(theta1=theta[1:DecisionVector[minOutput]],
                              theta2=Theta[1:DecisionVector[minOutput],k],
                              CLdec=CLdec[1:DecisionVector[minOutput]])
      }
    }
  }
  return(CUx)
}
```

D.1.3 Integrating out x^*

We present code for the case where our emulator is completely separable. This function assumes that all of the integrals involving x^* only have been computed. The inputs, *hxInt*,*hxhxtInt*,*rxInt*,*rxrxtInt*, and *hxrxtInt*, represent the five integrals

$$\int_{-1}^{1} h(x^*)p(x^*)dx^*, \qquad \int_{-1}^{1} h(x^*)h(x^*)^T p(x^*)dx^*,$$
$$\int_{-1}^{1} r^x(|x^*-\Omega|)p(x^*)dx^*, \qquad \int_{-1}^{1} r^x(|x^*-\Omega|)r^x(|x^*-\Omega|)^T p(x^*)dx^*,$$
$$\int_{-1}^{1} h(x^*)r^x(|x^*-\Omega|)p(x^*)dx^*,$$

required in the separable case and were derived in Appendix B for our example.

```
RequiredIntegrals <- function(theta,Theta,CLdec,DecisionVector,Rdec,V,SIGu,
                              hxInt,hxhxtInt,rxInt,rxrxtInt,hxrxtInt){
  kdec <- g(x=c(1,1,1),theta=theta)
  kdeckdect <- outer(kdec,kdec)
  gxInt <- kdec*hxInt
  gxgxTInt <- kdeckdect * hxhxtInt
  rlen <- length(rxInt)
  #rdec is 3 dimensional dim(rdec) = c(dim(V)[1],dim(V)[2],rlen)
  #If DecisionVector[min(i,j)]=0 then rdec[i,j,k] = 1 for all k
  #DecisionVector is strictly increasing in it's indices (it represents
  #temporal ordering of decisions)
  rdec <- array(1,dim=c(dim(V)[1],dim(V)[2],rlen))
  FirstDec <- NULL
  ind <- 1
```

```
while(is.null(FirstDec)){
  if(DecisionVector[ind]>0)
    FirstDec <- ind
  ind <- ind+1
}
for(i in FirstDec:dim(V)[1]){
  for(j in FirstDec:dim(V)[2]){
    for(k in 1:rlen){
      minOutput <- min(i,j)
      rdec[i,j,k] <- Rdec(theta1=theta[1:DecisionVector[minOutput]],
                          theta2=Theta[1:DecisionVector[minOutput],k],
                          CLdec=CLdec[1:DecisionVector[minOutput]])
    }
  }
}
covuUInt <- array(0,dim=dim(rdec))
covuUcovUuInt <- array(0,dim=c(dim(rdec),length(SIGu),rlen,length(SIGu)))
covugIntegral <- array(0,dim=c(length(kdec),dim(rdec)))
for(i in 1:length(SIGu)){
  for(k in 1:length(SIGu)){
    for(l in 1:rlen){
      covuUInt[i,k,l] <- V[i,k]*SIGu[i]*SIGu[k]*rdec[i,k,l]*rxInt[l]
      for(p in 1:length(kdec)){
        covugIntegral[p,i,k,l] <- V[i,k]*SIGu[i]*SIGu[k]*kdec[p]*
                                  rdec[i,k,l]*hxrxtInt[p,l]
      }
      for(m in 1:length(SIGu)){
        for(n in 1:rlen){
          for(j in 1:length(SIGu)){
            covuUcovUuInt[i,k,l,m,n,j] <- V[i,k]*SIGu[i]*SIGu[k]*
                                          V[m,j]*SIGu[m]*SIGu[j]*rdec[i,k,l]
                                          *rdec[m,j,n]*rxrxtInt[l,n]
          }
        }
      }
    }
  }
```

```
  }
  return(list(gxInt=gxInt,gxgxTInt=gxgxTInt,covuUInt=covuUInt,
              covuUcovUuInt=covuUcovUuInt,covugIntegral=covugIntegral))
}
```

D.1.4 Obtaining the beliefs about y

We have the five integrals required for calculating beliefs of the complex system. We need to now perform the correct array manipulations to obtain those beliefs. We do this for both the case where we may obtain robust inversion of $Var\left[F\right]$ using the cholesky decomposition, and for the case where we must resort to computing $Var\left[F\right]^{-1}$ directly using a generalized inverse.

```
ExpectRegression <- function(AdjEBetas,gxInt){
  #gxInt is an m vector, AdjEBetas is an nm matrix. Result is n vector
  m <- length(gxInt)
  library(tensor)
  tensor(AdjEBetas,gxInt,2,1)
}


ExpectResid <- function(Q,HalfUp,covuUInt,InvVDat,FminusEF){
  if(is.null(InvVDat)&&is.null(Q)){
    stop("Both InvVDat and Q are NULL")
  }
  if(is.null(InvVDat)){
    Qdim <- dim(Q)
    dim(Q) <- c(Qdim[1]*Qdim[2],Qdim[3]*Qdim[4])
    intDim <- dim(covuUInt)
    dim(covuUInt) <- c(intDim[1],intDim[2]*intDim[3])
    y <- backsolve(Q,t(covuUInt),transpose=TRUE)
    x <- HalfUp
    xdim <- dim(x)
    dim(x) <- c(xdim[1]*xdim[2],xdim[3]*xdim[4])
    EResid <- crossprod(y,x)
    dim(EResid) <- c(intDim[1])
    return(EResid)
```

```
  }
  else{
    library(tensor)
    RHSEResid <- tensor(InvVDat,FminusEF,c(3,4),c(1,2))
    stopifnot(all(dim(RHSEResid) == c(dim(InvVDat)[1],dim(InvVDat)[2])))
    intDim <- dim(covuUInt)
    EResid <- tensor(covuUInt,RHSEResid,c(2,3),c(1,2))
    stopifnot(dim(EResid) == intDim[1])
    return(EResid)
  }
}

#Now the four parts of the integral of the adjusted variance

IntAdjVBG <- function(AdjvarBetas,gxgxTInt){
  library(tensor)
  i <- dim(AdjvarBetas)[1]
  j <- dim(AdjvarBetas)[3]
  Ans <- tensor(AdjvarBetas,gxgxTInt,c(2,4),c(1,2))
  stopifnot(all(dim(Ans)==c(i,j)))
  Ans
}

IntAdjux <- function(Q,V,SIGu,covuUcovUuInt,InvVDat){
  if(is.null(Q)&&is.null(InvVDat))
    stop("Both Q and InvVDat NULL")
  library(tensor)
  A <- array(0,dim(V))
  for(i in 1:length(V[,1])){
    for(j in 1:length(V[1,])){
      A[i,j] <- V[i,j]*SIGu[i]*SIGu[j]
    }
  }
  if(is.null(InvVDat)){
    Qdim <- dim(Q)
    dim(Q) <- c(Qdim[1]*Qdim[2],Qdim[3]*Qdim[4])
    InvDat <- chol2inv(Q)
```

```
    dim(InvDat) <- Qdim
    B <- tensor(InvDat,covuUcovUuInt,c(1,2,3,4),c(2,3,4,5))
    return(A - B)
  }
  else{
    B <- tensor(InvVDat,covuUcovUuInt,c(1,2,3,4),c(2,3,4,5))
    return(A - B)
  }
}

IntCovsBthenU <- function(covugIntegral,VarBeta,Q,G,InvVDat){
  #dim(covugIntegral) is c(length(g),dim(cov(u(x,theta),U))
  if(is.null(Q) && is.null(InvVDat)){
    stop("Both Q and InvVDat are NULL")
  }
  library(tensor)
  VarBG <- tensor(VarBeta,G,4,1)
  if(is.null(InvVDat)){
    Qdim <- dim(Q)
    dim(Q) <- c(Qdim[1]*Qdim[2],Qdim[3]*Qdim[4])
    VBdim <- dim(VarBG)
    dim(VarBG) <- c(VBdim[1]*VBdim[2],VBdim[3]*VBdim[4])
    y <- backsolve(Q,t(VarBG),transpose=TRUE)
    covugIntegral <- aperm(covugIntegral,c(3,4,1,2))
    covdim <- dim(covugIntegral)
    dim(covugIntegral) <- c(covdim[1]*covdim[2],covdim[3]*covdim[4])
    x <- backsolve(Q,covugIntegral,transpose=TRUE)
    dim(y) <- c(Qdim[3]*Qdim[4],VBdim[1],VBdim[2])
    dim(x) <- c(Qdim[1]*Qdim[2],covdim[3],covdim[4])
    ans <- tensor(y,x,c(1,3),c(1,2))
    stopifnot(all(dim(ans)==c(VBdim[1],covdim[4])))
    return(ans)
  }
  else{
    covugIntegral <- aperm(covugIntegral,c(3,4,1,2))
    covdim <- dim(covugIntegral)
    VBdim <- dim(VarBG)
```

```
    RHSofManipulation <- tensor(InvVDat,covugIntegral,c(3,4),c(1,2))
    answer <- tensor(VarBG,RHSofManipulation,c(2,3,4),c(3,1,2))
    stopifnot(all(dim(answer)==c(VBdim[1],covdim[4])))
    return(answer)
  }
}

IntCovsUthenB <- function(covugIntegral,VarBeta,Q,G,InvVDat){
  if(is.null(Q) && is.null(InvVDat))
    stop("Q and InvVDat are both NULL")
  library(tensor)
  VarBG <- tensor(VarBeta,G,2,1)
  VarBG <- aperm(VarBG,c(1,4,2,3))
  VBdim <- dim(VarBG)
  covdim <- dim(covugIntegral)
  if(is.null(InvVDat)){
    Qdim <- dim(Q)
    dim(Q) <- c(Qdim[1]*Qdim[2],Qdim[3]*Qdim[4])
    dim(VarBG) <- c(VBdim[1]*VBdim[2],VBdim[3]*VBdim[4])
    dim(covugIntegral) <- c(covdim[1]*covdim[2],covdim[3]*covdim[4])
    y <- backsolve(Q,t(covugIntegral),transpose=TRUE)
    x <- backsolve(Q,VarBG,transpose=TRUE)
    dim(y) <- c(Qdim[3]*Qdim[4],covdim[1],covdim[2])
    dim(x) <- c(Qdim[1]*Qdim[2],VBdim[3],VBdim[4])
    ans <- tensor(y,x,c(1,2),c(1,3))
    stopifnot(all(dim(ans)==c(covdim[2],VBdim[3])))
    return(ans)
  }
  else{
    RHSofManipulation <- tensor(InvVDat,VarBG,c(3,4),c(1,2))
    answer <- tensor(covugIntegral,RHSofManipulation,c(1,3,4),c(4,1,2))
    stopifnot(all(dim(answer)==c(covdim[2],VBdim[3])))
    return(answer)
  }
}

#Now the four integrals of Int[E_F[f(x*,theta)]E_F[f(x*,theta)]^Tp(x*)dx*]
```

```
IntEbgEbg <- function(gxgxTInt,AdjEBetas){
  library(tensor)
  temp <- tensor(AdjEBetas,gxgxTInt,2,1)
  tensor(AdjEBetas,temp,2,2)
}

IntEbgEux <- function(covugIntegral,AdjEBetas,HalfUp,Q,InvVDat,FminusEF){
  if(is.null(Q)&&is.null(InvVDat))
    stop("Q and InvVDat both NULL")
  if(is.null(InvVDat)){
    x <- HalfUp
    library(tensor)
    LHS <- tensor(AdjEBetas,covugIntegral,2,1)
    Qdim <- dim(Q)
    dim(Q) <- c(Qdim[1]*Qdim[2],Qdim[3]*Qdim[4])
    LHSdim <- dim(LHS)
    dim(LHS) <- c(LHSdim[1]*LHSdim[2],LHSdim[3]*LHSdim[4])
    y <- backsolve(Q,t(LHS),transpose=TRUE)
    ans <- crossprod(y,x)
    dim(ans) <- c(LHSdim[1],LHSdim[2])
    return(ans)
  }
  else{
    library(tensor)
    LHS <- tensor(AdjEBetas,covugIntegral,2,1)
    LHSdim <- dim(LHS)
    RHS <- tensor(InvVDat,FminusEF,c(3,4),c(1,2))
    ans <- tensor(LHS,RHS,c(3,4),c(1,2))
    stopifnot(all(dim(ans)==c(LHSdim[1],LHSdim[2])))
    return(ans)
  }
}

IntEuxEux <- function(covuUcovUuInt,HalfUp,Q,InvVDat,FminusEF){
  if(is.null(Q)&&is.null(InvVDat))
    stop("Q and InvVDat both NULL")
```

```
  if(is.null(InvVDat)){
    library(tensor)
    Qdim <- dim(Q)
    dim(Q) <- c(Qdim[1]*Qdim[2],Qdim[3]*Qdim[4])
    x <- HalfUp
    covuUcovuUInt <- aperm(covuUcovUuInt,c(1,2,3,6,4,5))
    Idim <- dim(covuUcovuUInt)
    dim(covuUcovuUInt) <- c(Idim[1]*Idim[2]*Idim[3]*Idim[4],Idim[5]*Idim[6])
    y <- backsolve(Q,t(covuUcovuUInt),transpose=TRUE)
    Ans <- crossprod(y,x)
    dim(Ans) <- c(Idim[1:4])
    Ans <- aperm(Ans,c(1,4,2,3))
    ansdim <- dim(Ans)
    dim(Ans) <- c(ansdim[1]*ansdim[2],ansdim[3]*ansdim[4])
    y2 <- backsolve(Q,t(Ans),transpose=TRUE)
    final <- crossprod(y2,x)
    dim(final) <- c(ansdim[1],ansdim[2])
    return(final)
  }
  else{
    library(tensor)
    Left <- tensor(InvVDat,FminusEF,c(3,4),c(1,2))
    Right <- Left
    covuUcovuUInt <- aperm(covuUcovUuInt,c(1,2,3,6,4,5))
    RHSofManipulation <- tensor(covuUcovuUInt,Right,c(5,6),c(1,2))
    Answer <- tensor(Left,RHSofManipulation,c(1,2),c(2,3))
    stopifnot(all(dim(Answer)==c(dim(covuUcovuUInt)[1],dim(covuUcovuUInt)[4])))
    return(Answer)
  }
}

ExpectY <- function(ERegress,EResid){
  ERegress+EResid
}

VarY <- function(EY,SIGdis,intadjvbg,intadjux,intcovsbthenu,intcovsuthenb,
                 intebgebg,intebgeux,inteuxeux){
```

```
  muSquared <- outer(EY,EY)
  EVarfx <- intadjvbg + intadjux - intcovsbthenu - intcovsuthenb
  VarEfx <- intebgebg + inteuxeux + intebgeux + t(intebgeux) - muSquared
  VarEfx+EVarfx+SIGdis
}
```

D.1.5 Decision-dependent forecast

Based on the system beliefs and observations, we present code to adjust our beliefs about the system in order to obtain decision dependent forecasts for the case where we only have one intervention point.

```
Forecast <- function(Ey,Vy,SIGe,hist,intervention,future,Zh,Zt=NULL,
                     is.intervention=FALSE){
  #hist,intervention and future are integers describing the number of
  #quantities measured at the three different time points.
  #hist+intervention+future=length(Ey)
  #e_h and e_t are assumed uncorrelated and to have the same variance.
  #Thereofre we have SIGe=Var[e_h]=Var[e_t]
  #This would need altering if we had an error matrix
  if(!(length(Ey)==hist+intervention+future))
    stop("Number of elements specified does not match number in state vector")
  if(!(length(Zh)==hist))
    stop("Number historical observations is not as specified")
  if(!is.null(Zt)){
    if(!(length(Zt)==intervention))
      stop("Number of time t points observed is not as specified")
  }
  #Forecasting future observations at the intervention point
  if(is.intervention){
    EZt <- Ey[(hist+1):(hist+intervention)]
    #Assume e_h and e_t are uncorrelated
    CovZtZh <- Vy[(hist+1):(hist+intervention),1:hist]
    if(hist >1){
      ObsErr <- diag(SIGe^2,nrow=hist,ncol=hist)
      VZh <- ObsErr + Vy[1:hist,1:hist]
    }
    else
```

```
    VZh <- SIGe^2 + Vy[1:hist,1:hist]
  EZh <- Ey[1:hist]
  diff <- Zh - EZh
  Qz <- chol(VZh)
  x <- backsolve(Qz,diff,transpose=TRUE)
  y <- backsolve(Qz,t(CovZtZh),transpose=TRUE)
  addition <- crossprod(y,x)
  dim(addition) <- dim(EZt)
  ExpectFor <- EZt + addition
  Vyt <- Vy[(hist+1):(hist+intervention),(hist+1):(hist+intervention)]
  if(intervention>1){
    ObsErrt <- diag(SIGe^2,nrow=intervention,ncol=intervention)
    VZt <- ObsErrt + Vyt
  }
  else
    VZt <- SIGe^2 + Vyt
  VarFor <- VZt - crossprod(y)
  return(list(Expectation=ExpectFor,Variance=VarFor))
}
#Forecasting the state of the system given observations at both
#time points.
else{
  Ezhzt <- Ey[1:(hist+intervention)]
  Eyhytyf <- Ey
  data <- c(Zh,Zt)
  diff <- data-Ezhzt
  obsErr <- diag(SIGe^2,nrow=(hist+intervention),ncol=(hist+intervention))
  Vzhzt <- obsErr + Vy[1:(hist+intervention),1:(hist+intervention)]
  Covyhytyf.zhzt <- Vy[,1:(hist+intervention)]
  Qz <- chol(Vzhzt)
  x <- backsolve(Qz,diff,transpose=TRUE)
  y <- backsolve(Qz,t(Covyhytyf.zhzt),transpose=TRUE)
  dim(Eyhytyf) <- c(length(Ey),1)
  ExpectFor <- Eyhytyf + crossprod(y,x)
  VarFor <- Vy - crossprod(y)
  return(list(Expectation=ExpectFor, Variance=VarFor))
}
```

```
}
```

D.1.6 Wrappers

Here are the required wrappers needed in order to compute a forecast using our prior beliefs about the emulator quantities, runs on the model, and observations of the complex system.

```
#First we update the regression coefficients and calculate other non
#decision-dependent quantities that we'll need to recycle many times.
OneOffs <- function(X,Theta,EBetas,VarBetas,Runs,CLx,CLdec,SIGu,V,Rx,Rdec,
                    CholWorks=TRUE,DecisionVector){
  VarRuns <- VarF(X=X,Theta=Theta,VarBeta=VarBetas,Rx=Rx,Rdec=Rdec,CLx=CLx,
                 CLdec=CLdec,SIGu=SIGu,V=V,DecisionVector)
  FminusEF <- Fdiff(EBeta=EBetas,X=X,Theta=Theta,Data=Runs)
  #Here we try to compute the choelsky factors. If we can do this all of our
  #forecast calculations will be made using robust inversion of the data
  #array. If not, we compute the generalized inverse and use the tensor
  #function.
  Q <- try(tCholFactors(VarRuns),silent=TRUE)
  if(inherits(Q,"try-error"))
    CholWorks <- FALSE
  if(CholWorks){
    HalfUp <- HalfUpdate(Q=Q,FminusEF=FminusEF)
    AEBetas <- AdjEBetas(EBeta=EBetas,Q=Q,X=X,Theta=Theta,HalfUp=HalfUp,
                         VarBeta=VarBetas,InvVDat=NULL,FminusEF=FminusEF)
    AVBetas <- AdjVarBetas(VarBeta=VarBetas,Q=Q,X=X,Theta=Theta,InvVDat=NULL)
    return(list(VarRuns=VarRuns,Q=Q,HalfUp=HalfUp,AEBetas=AEBetas,
                AVBetas=AVBetas,InvVDat=NULL,FminusEF=FminusEF))
  }
  else{
    library(MASS)
    Vdim = dim(VarRuns)
    dim(VarRuns) <- c(Vdim[1]*Vdim[2],Vdim[3]*Vdim[4])
    InvVar <- ginv(VarRuns)
    dim(InvVar) <- c(Vdim[1],Vdim[2],Vdim[3],Vdim[4])
    Q <- NULL
    HalfUp <- NULL
```

```
    AEBetas <- AdjEBetas(EBeta=EBetas,Q=Q,X=X,Theta=Theta,HalfUp=HalfUp,
                         VarBeta=VarBetas,InvVDat=InvVar,FminusEF=FminusEF)
    AVBetas <- AdjVarBetas(VarBeta=VarBetas,Q=Q,X=X,Theta=Theta,InvVDat=InvVar)
    return(list(VarRuns=VarRuns,Q=Q,HalfUp=HalfUp,AEBetas=AEBetas,
                AVBetas=AVBetas,InvVDat=InvVar,FminusEF=FminusEF))
  }
}

SeparableForecast <- function(theta,VarRuns,Q,HalfUp,AEBetas,AVBetas,X,Theta,
EBetas,VarBetas,Runs,Zh,Zt=NULL,SIGdis,SIGe,SIGu,V,CLx,CLdec,Rx,Rdec,hist,
future,intervention,is.intervention=FALSE,InvVDat,FminusEF,DecisionVector,
hxInt,hxhxtInt,rxInt,rxrxtInt,hxrxtInt){
  reqs <- RequiredIntegrals(theta=theta,Theta=Theta,CLdec=CLdec,
DecisionVector=DecisionVector,Rdec=Rdec,V=V,SIGu=SIGu,hxInt=hxInt,
hxhxtInt=hxhxtInt,rxInt=rxInt,rxrxtInt=rxrxtInt,hxrxtInt=hxrxtInt)
  gInts <- list(gxInt=reqs$gxInt,gxgxTInt=reqs$gxgxTInt)
  covInts <- list(covuUInt=reqs$covuUInt,covuUcovUuInt=reqs$covuUcovUuInt)
  gcovsInt <- reqs$covugIntegral
  ourG <- G(X,Theta)
  Beliefs <- Job3(gInts=gInts,covInts=covInts,gcovsInt=gcovsInt,AEBetas=AEBetas,
Q=Q,HalfUp=HalfUp,SIGdis=SIGdis,AVBetas=AVBetas,V=V,SIGu=SIGu,VarBeta=VarBetas,
ourG=ourG,InvVDat=InvVDat,FminusEF=FminusEF)
  For <- Forecast(Ey=Beliefs$Ey,Vy=Beliefs$Vy,SIGe=SIGe,hist=hist, Zh=Zh,
intervention=intervention,future=future,Zt=Zt,is.intervention=is.intervention)
  return(list(Expectation=For$Expectation,Variance=For$Variance))
}

#This function is used to combine all of the array manipulations in order
#to obtain beliefs about the complex system.
Job3 <- function(gInts,covInts,gcovsInt,AEBetas,Q,HalfUp,SIGdis,AVBetas,V,
                 SIGu,VarBeta,ourG,InvVDat,FminusEF){
  ERegress <-  ExpectRegression(AdjEBetas=AEBetas,gxInt=gInts$gxInt)
  EResid <- ExpectResid(Q=Q,HalfUp=HalfUp,covuUInt=covInts$covuUInt,
                        InvVDat=InvVDat,FminusEF=FminusEF)
  intadjvbg <-  IntAdjVBG(AdjvarBetas=AVBetas,gxgxTInt=gInts$gxgxTInt)
  intadjux <-  IntAdjux(Q=Q,V=V,SIGu=SIGu,covuUcovUuInt=covInts$covuUcovUuInt,
                        InvVDat=InvVDat)
```

```
  intcovsbthenu <-  IntCovsBthenU(covugIntegral=gcovsInt,VarBeta=VarBeta,Q=Q,
                                  G=ourG,InvVDat=InvVDat)
  intcovsuthenb <-  IntCovsUthenB(covugIntegral=gcovsInt,VarBeta=VarBeta,Q=Q,
                                  G=ourG,InvVDat=InvVDat)
  intebgebg <-  IntEbgEbg(gxgxTInt=gInts$gxgxTInt,AdjEBetas=AEBetas)
  intebgeux <-  IntEbgEux(covugIntegral=gcovsInt,AdjEBetas=AEBetas,Q=Q
                          HalfUp=HalfUp,InvVDat=InvVDat,FminusEF=FminusEF)
  inteuxeux <-  IntEuxEux(covuUcovUuInt=covInts$covuUcovUuInt,Q=Q,
                          HalfUp=HalfUp,InvVDat=InvVDat,FminusEF=FminusEF)
  EY <- ExpectY(ERegress=ERegress,EResid=EResid)
  VY <- VarY(EY=EY,SIGdis=SIGdis,intadjvbg=intadjvbg,intadjux=intadjux,
             intcovsbthenu=intcovsbthenu,intcovsuthenb=intcovsuthenb,
             intebgebg=intebgebg,intebgeux=intebgeux,inteuxeux=inteuxeux)
  return(list(Ey=EY,Vy=VY))
}
```

D.2 Sequential Emulation

D.2.1 Sampling from log-normal distributions

Here we provide code for performing the Sequential Emulation algorithm using C-GOLDSTEIN and DICE in the way that we did for the example of chapter 4. First we provide code for evaluating our expected loss given a decision dependent forecast. This is done with respect to a lognormal distribution, and we provide functions for both the univariate and multivariate log-normal distributions here. Our approach is to first calculate the mean and variance of the Normal distribution corresponding to the log-normal distribution with mean and variance given by our forecast. We can use these quantities to sample from the log-normal distributions using standard functions in R.

The function *GetMuandSigma* converts a mean and variance from a log-normal distribution into the mean and variance of the corresponding normal distribution. The function *UniLogSamples* draws samples from a univariate log-normal distribution. The function *MultiLogSamples* draws samples from a multivariate log-normal distribution. The remaining functions in this section are used for calculating losses

from DICE for each value of the sample.

```
GetMuandSigma <- function(Ey,Vy){
  if(length(Ey)==1){
    SigSq <- log(1 + (Vy/(Ey^2)))
    Mu <- log(Ey) - (0.5*SigSq)
    Sigma <- SigSq
  }
  else{
    n <- length(Ey)
    stopifnot(all(dim(Vy)==c(n,n)))
    Mu <- rep(0,n)
    Sigma <- matrix(0,n,n)
    for(i in 1:n){
      Sigma[i,i] <- log(1+(Vy[i,i]/Ey[i]^2))
      Mu[i] <- log(Ey[i]) - 0.5*Sigma[i,i]
    }
    for(i in 1:n){
      for(j in 1:n){
        if(i != j)
          Sigma[i,j] <- log(1 + (Vy[i,j]*
                        exp((-1)*(Mu[i]+Mu[j]+Sigma[i,i]/2 + Sigma[j,j]/2))))
      }
    }
  }
  return(list(Mu=Mu,Sigma=Sigma))
}

UniLogSamples <- function(n,mean,variance){
  NormalMoments <- GetMuandSigma(Ey=mean,Vy=variance)
  rlnorm(n=n,meanlog=NormalMoments$Mu,sdlog=sqrt(NormalMoments$Sigma))
}

MultiLogSamples <- function(n,mean,variance){
  NormalMoments <- GetMuandSigma(Ey=mean,Vy=variance)
  xis <- rmvnorm(n=n,mean=NormalMoments$Mu,sigma=NormalMoments$Sigma)
  exp(xis)
}
```

```
MonteMLoss <- function(mean,variance,LossFun,numSamples,Thetas,rho0,gp,b10,
                       gb0,futureRate){
  samples <- MultiLogSamples(n=numSamples,mean=mean,variance=variance)
  Losses <- sapply(1:numSamples,function(i) LossFun(yh=samples[i,1],
                  yt=samples[i,2],yf=samples[i,3],theta=c(Thetas[1,1],
                  Thetas[1,2]),rho0=rho0,gp=gp,b10=b10,gb0=gb0,
                  futureRate=futureRate))
  return(sum(Losses)/numSamples)
}

thisDICE <- function(yh,yt,yf,theta,rho0,gp,gb0,b10,futureRate){
  stopifnot(length(theta)==2)
  #Put thetas onto [0.1,1] (from [-1,1])
  theta1 <- (theta[1]+1)/2
  theta1 <- (theta1*0.9) + 0.1
  theta2 <- (theta[2]+1)/2
  theta2 <- (theta2*0.9) +0.1
  (-1)*DICE(Yh=yh,Zt=yt, Yp= yf, IntTime=4,RateNow=theta1, RateInt=theta2,
            rho0=rho0,gp=gp,gb0=gb0,b10=b10,delb=delb,futureRate=futureRate)
}
```

D.2.2 Multi-level adjustment

Here we provide the key functions used for the multi-level update of our beliefs about expected loss. The code here represents the Bayes Linear adjustment presented in section 3.4. The function *BuildFine()* performs all of the one-off elements of the adjustment of the fine emulator that do not depend on the input of the emulator. Here, the coarse correlated residual is adjusted by all runs we have on the coarse, and the coefficients ρ and γ are updated by the difference between coarse and fine funs. The inputs are *Coarse*, which is a list containing the linear model we fitted to the coarse data; *Priors*, which is a list of the prior means and variances of the multi-level model coefficients, the variance functions for both the coarse and accurate correlated residuals, the variance of the uncorrelated error for both models, and the correlation parameters for both the coarse and accurate residual processes; D, which is a list

containing the design matrix of our coarse and accurate runs as well as matrices for both the coarse and accurate runs evaluated at those points; *CoarseData*, which is the design matrix for the coarse runs used to build the linear model contained in *Coarse*. The output of the function is a list containing the adjusted moments of each of the quantities mentioned as well as the matrices required for Bayes Linear adjustment of any quantities that depend on the inputs to the model. Having a function that does all of these jobs one time allows the emulators that we build to be evaluated as quickly as possible after this initial adjustment.

```
BuildFine <- function(Coarse,Priors,D,CoarseData){
  #We begin by updating the coarse residual at each design point.
  if(length(CoarseData[1,])>(length(D$X[1,])+1))
    stop("CoarseData must only have columns for the design and Eloss")
  m <- length(CoarseData$Eloss) + length(D$Fo)
  m1 <- length(CoarseData$Eloss)
  sumBg <- predict.lm(object=Coarse$model,newdata=as.data.frame(D$X))
  #Now we calculate E^c, which with random error we substitute in E_R^C[e^c(X)]
  #where X is the coarse/fine design and e^c(x) is the correlated part of
  #the coarse residual.
  BigResids <- resid(Coarse$model)
  JointRunCResids <- D$F1 - sumBg
  tData <- c(BigResids,JointRunCResids)
  CoarseData <- as.matrix(CoarseData)
  if(dim(D$X)[2]<2)
    totalXs <- as.matrix(data.frame(t1=c(CoarseData[,1],D$X[,1])))
  else
    totalXs <- rbind(CoarseData[,-(length(CoarseData[1,]))],D$X)
  #totalXs is the design matrix for all of the runs we have on the coarse.
  EtData <- sapply(1:m,function(i) Priors$Eec(totalXs[i,]))
  CoarseDiff <- tData - EtData
  VCoarse <- matrix(0,nrow=m,ncol=m)
  if(!length(Priors$CCorrLength)==length(D$X[1,])){
    if(length(Priors$CCorrLength)<2)
      Priors$CCorrLength <- rep(Priors$CCorrLength,length(D$X[1,]))
    else
      stop("Incorrectly specified vector of correlation parameters")
  }
```

```
CLengths <- diag(Priors$CCorrLength,nrow=length(D$X[1,]),ncol=length(D$X[1,]))
for(i in 1:m){
  for(j in 1:m){
    if(i > m1)
      a <- D$X[(i-m1),]
    else
      a <- CoarseData[i,-(length(CoarseData[1,]))]
    if(j > m1)
      b <- D$X[(j-m1),]
    else
      b <- CoarseData[j,-(length(CoarseData[1,]))]
    exponent <- t(a-b)%*%CLengths%*%(a-b)
    VCoarse[i,j] <- exp(-exponent)*sqrt(Priors$Vec(totalXs[i,]))
                        *sqrt(Priors$Vec(totalXs[j,]))
  }
}
#Now we add the coarse uncorrelated variance to the variance matrix
VdeltaCoarse <- diag(rep(Priors$vdc,m),nrow=m,ncol=m)
VCoarse <- VCoarse+VdeltaCoarse
#Try robust inversion via cholesky decomposition and use a generalized
#inverse if the eigenvalues are too small.
CQ <- try(chol(VCoarse),TRUE)
if(inherits(CQ,"try-error")){
  library(MASS)
  VCinv <- ginv(VCoarse)
  CQ <- NULL
  CHalfUp <- NULL
}
else{
  CHalfUp <- backsolve(CQ,CoarseDiff,transpose=TRUE)
  VCinv <- NULL
  CDiff <- NULL
}
#We call the function AdjustCoarseResids() below to adjust the coarse
#residuals, using the matrices we have constructed above.
Ec <- sapply(1:length(D$Fo), function(i) AdjustCoarseResids(x=D$X[i,],
             PriorResid=list(CLength=Priors$CCorrLength,Eec=Priors$Eec,
```

```
            Vec=Priors$Vec),CResidData=totalXs,Q=CQ,HalfUp=CHalfUp,
            VDinv=VCinv,Diff=CoarseDiff)$Eec)
##############################################
#We now adjust the parameters of the accurate model.
#We first construct the matrices and vectors required for the adjustment
#Difference between the data and it's expectation:
n <- length(D$Fo)
Ef <- sapply(1:n,function(i) Priors$Eef(x=D$X[i,]))
ED <- (Priors$ERho - 1)*sumBg + (Priors$EGamma - 1)*Ec + Ef
DelD <- D$F1 -D$Fo - ED
BgEc <- outer(sumBg,Ec)
EcBg <- t(BgEc)
#Variance of the data
VEf <- matrix(0,nrow=n,ncol=n)
if(!length(Priors$FCorrLength)==length(D$X[1,])){
  if(length(Priors$FCorrLength)<2)
    Priors$FCorrLength <- rep(Priors$FCorrLength,length(D$X[1,]))
  else
    stop("Incorrectly specified vector of correlation parameters")
}
FLengths <- diag(Priors$FCorrLength,nrow=length(D$X[1,]),ncol=length(D$X[1,]))
for(i in 1:n){
  for(j in 1:n){
    exponent <- t(D$X[i,]-D$X[j,])%*%FLengths%*%(D$X[i,]-D$X[j,])
    VEf[i,j] <- exp(-exponent)*sqrt(Priors$Vef(D$X[i,]))
                *sqrt(Priors$Vef(D$X[j,]))
  }
}
VDels <- diag(rep(Priors$vdf,n),nrow=n,ncol=n)
VD <- Priors$VRho*outer(sumBg,sumBg) + Priors$VGamma*outer(Ec,Ec)
                  + Priors$CRhoGamma*(BgEc+EcBg) + VEf + VDels
CovRhoD <- Priors$VRho*sumBg + Priors$CRhoGamma*Ec
CovGammaD <- Priors$CRhoGamma*sumBg + Priors$VGamma*Ec
#Try inverting the variance using the cholesky decomposition. Use
#A generalized inverse if the eigenvalues are too small.
Q <- try(chol(VD),TRUE)
if(inherits(Q,"try-error")){
```

```
    library(MASS)
    VDinv <- ginv(VD)
    EDRho <- Priors$ERho + CovRhoD%*%VDinv%*%DelD
    VDRho <- Priors$VRho - CovRhoD%*%VDinv%*%CovRhoD
    EDGamma <- Priors$EGamma + CovGammaD%*%VDinv%*%DelD
    VDGamma <- Priors$VGamma - CovGammaD%*%VDinv%*%CovGammaD
    CovDRhoGamma <- Priors$CRhoGamma - CovRhoD%*%VDinv%*%CovGammaD
    return(list(EDRho=EDRho,EDGamma=EDGamma,VDRho=VDRho,VDGamma=VDGamma,
                CDRhoGamma=CovDRhoGamma,CovRhoD=CovRhoD,CovGammaD=CovGammaD,
                Q=NULL,HalfUp=NULL,VDinv=VDinv,DelD=DelD,CQ=CQ,CHalfUp=CHalfUp,
                VCinv=VCoarseinv,CDiff=CoarseDiff,CoarseLocs=totalXs,Ec=Ec))
  }
  else{
    HalfUp <- backsolve(Q,DelD,transpose=TRUE)
    yrho <- backsolve(Q,CovRhoD,transpose=TRUE)
    EDRho <- Priors$ERho + crossprod(yrho, HalfUp)
    VDRho <- Priors$VRho - crossprod(yrho)
    ygam <- backsolve(Q,CovGammaD,transpose=TRUE)
    EDGamma <- Priors$EGamma + crossprod(ygam,HalfUp)
    VDGamma <- Priors$VGamma - crossprod(ygam)
    CovDRhoGamma <- Priors$CRhoGamma - crossprod(yrho,ygam)
    return(list(EDRho=EDRho,EDGamma=EDGamma,VDRho=VDRho,VDGamma=VDGamma,
                CDRhoGamma=CovDRhoGamma,CovRhoD=CovRhoD,CovGammaD=CovGammaD,
                Q=Q,HalfUp=HalfUp,VDinv=NULL,DelD=NULL,CQ=CQ,CHalfUp=CHalfUp,
                VCinv=VCoarseinv,CDiff=CoarseDiff,CoarseLocs=totalXs,Ec=Ec))
  }
}

AdjustCoarseResids <- function(x,PriorResid,CResidData,Q,VDinv,HalfUp,Diff){
  #CResidData is a matrix of locations of the observed coarse.
  #Only require E_D(e(x)), Var_D(e(x))
  if(length(x)!= length(CResidData[1,]))
     stop("x has different dimensions to the Design")
  #Step1 (the expensive bit). Create covariance
  if(!length(PriorResid$CLength)==length(x)){
    if(length(PriorResid$CLength)<2)
      PriorResid$CLength <- rep(PriorResid$CLength,length(x))
```

```
    else
      stop("Incorrectly specified vector of correlation parameters")
  }
  CLengths <- diag(PriorResid$CLength,nrow=length(x),ncol=length(x))
  n <- length(CResidData[,1])
  tcov <- sapply(1:n, function(i) (sqrt(PriorResid$Vec(x)))*
               (sqrt(PriorResid$Vec(CResidData[i,])))*(exp((-1)*
               ((x-CResidData[i,])%*%CLengths%*%(x-CResidData[i,]))))
  if(is.null(Q)){
    EDec <- PriorResid$Eec(x) + (tcov%*%VDinv%*%Diff)
    VDec <- PriorResid$Vec(x) - (tcov%*%VDinv%*%tcov)
  }
  else{
    yc <- backsolve(Q,tcov,transpose=TRUE)
    EDec <- PriorResid$Eec(x) + crossprod(yc,HalfUp)
    VDec <- PriorResid$Vec(x) - crossprod(yc)
  }
  return(list(Eec=EDec,Vec=VDec))
}
```

D.2.3 Obtaining coarse and accurate runs

The following functions obtain fast and accurate runs for each of the three emulators we wish to construct. The object *ComputerModelStuff* is a list containing all of the quantities required for forecasting as well as the elements from the output of *OneOffs*. In order to obtain the runs for emulating $A(\theta^1, z^1)$ we present the code for obtaining the coarse runs and the code used for obtaining the runs of both coarse and accurate models at the same locations through *AFast* and *AFull*. These functions are designed specifically to work for the example in chapter 4 and would need to be altered for a different problem.

```
AFast <- function(fastPoints,forecastgridpoints=NULL,ComputerModelStuff,
           Separable=TRUE,numSamples,rho0,gp,b10,gb0,futureRate){
  cms <- ComputerModelStuff
  #Create a data frame with theta1, theta2, zt, Eloss and use it to emulate.
  Eloss <- rep(0,length(fastPoints[,1]))
  for(i in 1:length(Eloss)){
```

```
    temp <- SeparableForecast(theta=c(fastPoints[i,1],fastPoints[i,2]),
         VarRuns=cms$VarRuns,Q=cms$Q,HalfUp=cms$HalfUp,AEBetas=cms$AEBetas,
         AVBetas=cms$AVBetas,X=cms$X,Theta=cms$Theta,EBetas=cms$EBetas,
         VarBetas=cms$VarBetas,Runs=cms$Runs,Zh=cms$Zh,Zt=fastPoints[i,3],
         SIGdis=cms$SIGdis,SIGe=cms$SIGe,SIGu=cms$SIGu,V=cms$V,CLx=cms$CLx,
         CLdec=cms$CLdec,Rx=cms$Rx,Rdec=cms$Rdec,hist=cms$hist,
         future=cms$future,intervention=cms$intervention,InvVDat=cms$InvVDat,
         FminusEF=cms$FminusEF,DecisionVector=cms$DecisionVector,
         hxhxtInt=cms$hxhxtInt,rxInt=cms$rxInt,rxrxtInt=cms$rxrxtInt,
         hxInt=cms$hxInt,hxrxtInt=cms$hxrxtInt)
    Eloss[i] <-  MonteMLoss(mean=temp$Expectation,variance=temp$Variance,
             LossFun=thisDICE,numSamples=numSamples,Thetas=fastPoints[i,-3],
             rho0=rho0,gp=gp,b10=b10,gb0=gb0,futureRate=futureRate)
  }
  data.frame(t1=fastPoints[,1],t2=fastPoints[,2],zt=fastPoints[,3],Eloss)
}

AFull <- function(FineDesign,ComputerModelStuff,numsamples,nfsamples=NULL,
                   rho0=0.03,gp=0.00257,gb0=-0.08,b10=0.03,futureRate=0.05){
  cms <- ComputerModelStuff
  ExpectCLoss <- c()
  ExpectFLoss <- c()
  for(i in 1:length(FineDesign[,1])){
    forc <- SeparableForecast(theta=c(FineDesign[i,1],FineDesign[i,2]),
          VarRuns=cms$VarRuns,Q=cms$Q,HalfUp=cms$HalfUp,AEBetas=cms$AEBetas,
          AVBetas=cms$AVBetas,X=cms$X,Theta=cms$Theta,EBetas=cms$EBetas,
          VarBetas=cms$VarBetas,Runs=cms$Runs,Zh=cms$Zh,Zt=FineDesign[i,3],
          SIGdis=cms$SIGdis,SIGe=cms$SIGe,SIGu=cms$SIGu,V=cms$V,CLx=cms$CLx,
          CLdec=cms$CLdec,Rx=cms$Rx,Rdec=cms$Rdec,hist=cms$hist,
          future=cms$future,intervention=cms$intervention,InvVDat=cms$InvVDat,
          FminusEF=cms$FminusEF,DecisionVector=cms$DecisionVector,
          hxInt=cms$hxInt,hxhxtInt=cms$hxhxtInt,rxInt=cms$rxInt,
          rxrxtInt=cms$rxrxtInt,hxrxtInt=cms$hxrxtInt)
     Elosses <- MonteMLoss(mean=forc$Expectation,variance=forc$Variance,
             LossFun=thisDICE,numSamples=numsamples,Thetas=FineDesign[i,-3],
             rho0=rho0,gp=gp,b10=b10,gb0=gb0,futureRate=futureRate)
     ExpectCLoss <- c(ExpectCLoss,Elosses)
```

```
    Elosses <- MonteMLoss(mean=forc$Expectation,variance=forc$Variance,
            LossFun=thisDICE,numSamples=nfsamples,Thetas=FineDesign[i,-3],
                     rho0=rho0,gp=gp,b10=b10,gb0=gb0,futureRate=futureRate)
    ExpectFLoss <- c(ExpectFLoss,Elosses)
  }
 data.frame(t1=FineDesign[,1],t2=FineDesign[,2],zt=FineDesign[,3],
            ECLoss=ExpectCLoss,EFLoss=ExpectFLoss)
}
```

As the code for generating the coarse runs is very similar to that used for generating the coarse and accurate runs at the same locations, we only include code of the latter type here for the remaining two emulators. In order to emulate $B^1_{\lambda^1}(\theta_{t_0}, z^1)$ we must find the minimum of our emulator for $A(\theta^1, z^1)$. This is done using the function *t2Strategy* which appears below and is called by *BFull* here.

```
BFull <- function(FDesign,AEmulator,ComputerModelStuff,numsamples,
                    nfsamples,rho0,gp,gb0,b10,futureRate){
  cms <- ComputerModelStuff
  ExpectedCLoss <- c()
  ExpectedFLoss <- c()
  t2 <- c()
  for(i in 1:length(FineDesign[,1])){
    temp <- t2Strategy(t1=FDesign$t1[i],zt=FDesign$zt[i],AEmulator=AEmulator)
    if(temp$t2 < -1)
      stop("min t2 should be greater than -1")
    if(temp$t2 >1)
      stop("min t2 should be less than 1")
    t2 <- c(t2,temp$t2)
    t1Coarse <- SeparableForecast(theta=c(FDesign$t1[i],temp$t2),
            VarRuns=cms$VarRuns,Q=cms$Q,HalfUp=cms$HalfUp,AEBetas=cms$AEBetas,
            AVBetas=cms$AVBetas,X=cms$X,Theta=cms$Theta,EBetas=cms$EBetas,
            VarBetas=cms$VarBetas,Runs=cms$Runs,Zh=cms$Zh,Zt=FDesign$zt[i],
            SIGdis=cms$SIGdis,SIGe=cms$SIGe,SIGu=cms$SIGu,V=cms$V,CLx=cms$CLx,
            CLdec=cms$CLdec,Rx=cms$Rx,Rdec=cms$Rdec,hist=cms$hist,
            future=cms$future,intervention=cms$intervention,InvVDat=cms$InvVDat,
            FminusEF=cms$FminusEF,DecisionVector=cms$DecisionVector,
            hxInt=cms$hxInt,hxhxtInt=cms$hxhxtInt,rxInt=cms$rxInt,
            rxrxtInt=cms$rxrxtInt,hxrxtInt=cms$hxrxtInt)
```

```
      ts <- as.matrix(c(FDesign$t1[i],temp$t2))
      dim(ts) <- c(1,2)
      Elosses <- MonteMLoss(mean=t1Coarse$Expectation,variance=t1Coarse$Variance,
              LossFun=thisDICE,numSamples=numsamples,Thetas=ts,rho0=rho0,gp=gp,
              b10=b10,gb0=gb0,futureRate=futureRate)
      ExpectCLoss <- c(ExpectCLoss,Elosses)
      Elosses <- MonteMLoss(mean=t1Coarse$Expectation,variance=t1Coarse$Variance,
              LossFun=thisDICE,numSamples=nfsamples,Thetas=ts,rho0=rho0,gp=gp,
              b10=b10,gb0=gb0,futureRate=futureRate)
      ExpectFLoss <- c(ExpectFLoss,Elosses)
    }
  data.frame(t1=FDesign$t1,zt=FDesign$zt,t2=t2,ECLoss=ExpectCLoss,
              EFLoss=ExpectFLoss)
}
```

The emulator for $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$ is constructed using runs obtained by running the functions *CFast* and *CFull*, the latter of which is below. Each expected loss is an evaluation of our emulator for $B^1_{\lambda^1}(\theta_{t_0}, z^1)$, which is called via the function *BFine*, below.

```
CFull <- function(Design,ComputerModelStuff,numSamples,nfsamples,BEmulator){
  cms <- ComputerModelStuff
  ExpectCLoss <- c()
  ExpectFLoss <- c()
  for(i in 1:length(Design$t1)){
    t1Coarse <-  SeparableForecast(theta=c(Design$t1[i],0),VarRuns=cms$VarRuns,
              Q=cms$Q,HalfUp=cms$HalfUp,AEBetas=cms$AEBetas,AVBetas=cms$AVBetas,
              X=cms$X,Theta=cms$Theta,EBetas=cms$EBetas,VarBetas=cms$VarBetas,
              Runs=cms$Runs,Zh=cms$Zh,Zt=NULL,SIGdis=cms$SIGdis,SIGe=cms$SIGe,
              SIGu=cms$SIGu,V=cms$V,CLx=cms$CLx,CLdec=cms$CLdec,Rx=cms$Rx,
              Rdec=cms$Rdec,hist=cms$hist,future=cms$future,hxInt=cms$hxInt,
              intervention=cms$intervention,is.intervention=TRUE,rxInt=cms$rxInt,
              InvVDat=cms$InvVDat,FminusEF=cms$FminusEF,hxhxtInt=cms$hxhxtInt,
              DecisionVector=cms$DecisionVector,rxrxtInt=cms$rxrxtInt,
              hxrxtInt=cms$hxrxtInt)
    samples <- UniLogSamples(n=numSamples,mean=t1Coarse$Expectation,
            variance=t1Coarse$Variance)
    Elosses <- sapply(1:length(samples),function(j) BFine(Design$t1[i],
```

```
            samples[j],Coarse=BEmulator$Coarse,BuiltFine=BEmulator$BuiltFine,
            Priors=BEmulator$Priors,D=BEmulator$D)$Ef1)
    ExpectCLoss <- c(ExpectCLoss,mean(Elosses))
    samples <- UniLogSamples(n=nfsamples,mean=t1Coarse$Expectation,
            variance=t1Coarse$Variance)
    Elosses <- sapply(1:nfsamples,function(j) BFine(Design$t1[i],samples[j],
            Coarse=BEmulator$Coarse,BuiltFine=BEmulator$BuiltFine,
            Priors=BEmulator$Priors,D=BEmulator$D)$Ef1)
    ExpectFLoss <- c(ExpectFLoss,mean(Elosses))
   }
  data.frame(t1=Design$t1,ECloss=ExpectCLoss,EFloss=ExpectFLoss)
 }
```

D.2.4 The Final Emulators

The following functions represent the emulators for each of the expected losses on our decision tree. We also include the function *t2Strategy*, which locates the minimum of our emulator for $A(\theta^1, z^1)$.

```
AFine <- function(t2,t1,zt,Coarse,BuiltFine,Priors,D){
  n <- length(D$X[,1])
  FLengths <- diag(Priors$FCorrLength,nrow=length(D$X[1,]),ncol=length(D$X[1,]))
  covefEF <- sapply(1:n,function(i) (sqrt(Priors$Vef(c(t1,t2,zt))))*
          (sqrt(Priors$Vef(D$X[i,])))*(exp((-1)*(t(c(t1,t2,zt)-D$X[i,])
          %*%FLengths%*%(c(t1,t2,zt)-D$X[i,])))))
  CResidUp <- AdjustCoarseResids(x=c(t1,t2,zt),PriorResid=list(
           CLength=Priors$CCorrLength,Eec=Priors$Eec,Vec=Priors$Vec),
           CResidData=BuiltFine$CoarseLocs,Q=BuiltFine$CQ,
           HalfUp=BuiltFine$CHalfUp,VDinv=BuiltFine$VCinv,
           Diff=BuiltFine$CDiff)
  sumBgx <- predict.lm(object=Coarse$model,newdata=data.frame(t1=t1,
                      t2=t2,zt=zt))
  if(is.null(BuiltFine$Q)){
    VDinv <- BuiltFine$VDinv
    DelD <- BuiltFine$DelD
    EDefx <- Priors$Eef(x=c(t1,t2,zt)) + (covefEF%*%VDinv%*%DelD)
    VDefx <- Priors$Vef(x=c(t1,t2,zt)) - (covefEF%*%VDinv%*%covefEF)
    CDefxRho <- Priors$CefRho(x=c(t1,t2,zt)) -
```

```
                (covefEF%*%VDinv%*%(BuiltFine$CovRhoD))
    CDefxGamma <- Priors$CefGamma(x=c(t1,t2,zt)) -
                  (covefEF%*%VDinv%*%(BuiltFine$CovGammaD))
  }
  else{
    Q <- BuiltFine$Q
    yef <- backsolve(Q,covefEF,transpose=TRUE)
    yRho <- backsolve(Q,BuiltFine$CovRhoD,transpose=TRUE)
    yGam <- backsolve(Q,BuiltFine$CovGammaD,transpose=TRUE)
    EDefx <- Priors$Eef(x=c(t1,t2,zt)) + crossprod(yef,BuiltFine$HalfUp)
    VDefx <- Priors$Vef(x=c(t1,t2,zt)) - crossprod(yef)
    CDefxRho <- Priors$CefRho(x=c(t1,t2,zt)) - crossprod(yef,yRho)
    CDefxGamma <- Priors$CefGamma(x=c(t1,t2,zt)) - crossprod(yef,yGam)
  }
  Ef1 <- (sumBgx*BuiltFine$EDRho)+((CResidUp$Eec)*(BuiltFine$EDGamma))+EDefx
  Vf1 <- BuiltFine$VDRho*(sumBgx^2))+(BuiltFine$VDGamma*(CResidUp$Eec^2))+
      (VDefx)+(2*BuiltFine$CDRhoGamma*sumBgx*CResidUp$Eec)+ Priors$vdf +
      (2*CDefxGamma*CResidUp$Eec)+(2*CDefxRho*sumBgx)+
      (CResidUp$Vec*(Priors$VGamma + (Priors$EGamma^2)))
  return(list(Ef1=Ef1,Vf1=Vf1))
}

#The next function is necessary only to run an optimiser over t2
#This is because of the way optimiser works in R
ExpectAFine <- function(t2,t1,zt,AEmulator){
  F <- AFine(t2=t2,t1=t1,zt=zt,Coarse=AEmulator$Coarse,D=AEmulator$D,
             BuiltFine=AEmulator$BuiltFine,Priors=AEmulator$Priors)
  F$Ef1
}

t2Strategy <- function(t1,zt,AEmulator){
  tAEmulator=AEmulator
  optimizer <- optimize(f=ExpectAFine,interval=c(-1,1),t1=t1,zt=zt,
                        AEmulator=tAEmulator)
  optimizer1 <- optimize(f=ExpectAFine,interval=c(-1,0),t1=t1,zt=zt,
                         AEmulator=tAEmulator)
  if(optimizer1$objective < optimizer$objective)
```

```
    optimizer <- optimizer1
  optimizer2 <- optimize(f=ExpectAFine,interval=c(0,1),t1=t1,zt=zt,
                         AEmulator=tAEmulator)
  if(optimizer2$objective < optimizer$objective)
    optimizer <- optimizer2
  t2=optimizer$minimum
  ELoss=optimizer$objective
  return(list(ELoss=ELoss,t2=t2))
}

BFine <- function(t1,zt,Coarse,BuiltFine,Priors,D){
  n <- length(D$X[,1])
  FLengths <- diag(Priors$FCorrLength,nrow=length(D$X[1,]),
                   ncol=length(D$X[1,]))
  covefEF <- sapply(1:n,function(i) (sqrt(Priors$Vef(c(t1,zt))))*
          (sqrt(Priors$Vef(D$X[i,])))*(exp((-1)*(t(c(t1,zt)-D$X[i,])
          %*%FLengths%*%(c(t1,zt)-D$X[i,]))))
  CResidUp <- AdjustCoarseResids(x=c(t1,zt),PriorResid=list(
           CLength=Priors$CCorrLength,Eec=Priors$Eec,Vec=Priors$Vec),
           CResidData=BuiltFine$CoarseLocs,Q=BuiltFine$CQ,
           HalfUp=BuiltFine$CHalfUp,VDinv=BuiltFine$VCinv,
           Diff=BuiltFine$CDiff)
  sumBgx <- predict.lm(object=Coarse$model,newdata=data.frame(t1=t1,zt=zt))
  if(is.null(BuiltFine$Q)){
    VDinv <- BuiltFine$VDinv
    DelD <- BuiltFine$DelD
    EDefx <- Priors$Eef(x=c(t1,zt)) + (covefEF%*%VDinv%*%DelD)
    VDefx <- Priors$Vef(x=c(t1,zt)) - (covefEF%*%VDinv%*%covefEF)
    CDefxRho <- Priors$CefRho(x=c(t1,zt)) -
                (covefEF%*%VDinv%*%(BuiltFine$CovRhoD))
    CDefxGamma <- Priors$CefGamma(x=c(t1,zt)) -
                  (covefEF%*%VDinv%*%(BuiltFine$CovGammaD))
  }
  else{
    Q <- BuiltFine$Q
    yef <- backsolve(Q,covefEF,transpose=TRUE)
    yRho <- backsolve(Q,BuiltFine$CovRhoD,transpose=TRUE)
```

```
    yGam <- backsolve(Q,BuiltFine$CovGammaD,transpose=TRUE)
    EDefx <- Priors$Eef(x=c(t1,zt)) + crossprod(yef,BuiltFine$HalfUp)
    VDefx <- Priors$Vef(x=c(t1,zt)) - crossprod(yef)
    CDefxRho <- Priors$CefRho(x=c(t1,zt)) - crossprod(yef,yRho)
    CDefxGamma <- Priors$CefGamma(x=c(t1,zt)) - crossprod(yef,yGam)
  }
  Ef1 <- (sumBgx*BuiltFine$EDRho) + ((CResidUp$Eec)*(BuiltFine$EDGamma)) + EDefx
  Vf1 <- (BuiltFine$VDRho*(sumBgx^2))+(BuiltFine$VDGamma*(CResidUp$Eec^2))+
      (VDefx)+(2*BuiltFine$CDRhoGamma*sumBgx*CResidUp$Eec)+ Priors$vdf
      (2*CDefxGamma*CResidUp$Eec)+(2*CDefxRho*sumBgx)+
      (CResidUp$Vec*(Priors$VGamma + (Priors$EGamma^2)))
  return(list(Ef1=Ef1,Vf1=Vf1))
}

CFine <- function(t1,Coarse,BuiltFine,Priors,D){
  n <- length(D$X[,1])
  FLengths <- diag(Priors$FCorrLength,nrow=length(D$X[1,]),
                   ncol=length(D$X[1,]))
  covefEF <- sapply(1:n,function(i) (sqrt(Priors$Vef(t1)))*
          (sqrt(Priors$Vef(D$X[i,])))*(exp((-1)*(t(c(t1)-D$X[i,])
          %*%FLengths%*%(c(t1)-D$X[i,])))))
  covefEF <- covefEF*Priors$Vef(1)
  CResidUp <- AdjustCoarseResids(x=c(t1),PriorResid=list(
           CLength=Priors$CCorrLength,Eec=Priors$Eec,Vec=Priors$Vec),
           CResidData=BuiltFine$CoarseLocs,Q=BuiltFine$CQ,
           HalfUp=BuiltFine$CHalfUp,VDinv=BuiltFine$VCinv,
           Diff=BuiltFine$CDiff)
  sumBgx <- predict.lm(object=Coarse$model,newdata=data.frame(t1=t1))
  if(is.null(BuiltFine$Q)){
    VDinv <- BuiltFine$VDinv
    DelD <- BuiltFine$DelD
    EDefx <- Priors$Eef(x=c(t1)) + (covefEF%*%VDinv%*%DelD)
    VDefx <- Priors$Vef(x=c(t1)) - (covefEF%*%VDinv%*%covefEF)
    CDefxRho <- Priors$CefRho(x=c(t1)) - (covefEF%*%VDinv%*%(BuiltFine$CovRhoD))
    CDefxGamma <- Priors$CefGamma(x=c(t1)) -
                  (covefEF%*%VDinv%*%(BuiltFine$CovGammaD))
  }
```

```
  else{
    Q <- BuiltFine$Q
    yef <- backsolve(Q,covefEF,transpose=TRUE)
    yRho <- backsolve(Q,BuiltFine$CovRhoD,transpose=TRUE)
    yGam <- backsolve(Q,BuiltFine$CovGammaD,transpose=TRUE)
    EDefx <- Priors$Eef(x=c(t1)) + crossprod(yef,BuiltFine$HalfUp)
    VDefx <- Priors$Vef(x=c(t1)) - crossprod(yef)
    CDefxRho <- Priors$CefRho(x=c(t1)) - crossprod(yef,yRho)
    CDefxGamma <- Priors$CefGamma(x=c(t1)) - crossprod(yef,yGam)
  }
  Ef1 <- (sumBgx*BuiltFine$EDRho) + ((CResidUp$Eec)*(BuiltFine$EDGamma)) + EDefx
  Vf1 <- BuiltFine$VDRho*(sumBgx^2))+(BuiltFine$VDGamma*(CResidUp$Eec^2))+
      (VDefx)+(2*BuiltFine$CDRhoGamma*sumBgx*CResidUp$Eec)+
      (2*CDefxGamma*CResidUp$Eec)+(2*CDefxRho*sumBgx)+
      (CResidUp$Vec*(Priors$VGamma + (Priors$EGamma^2))) + Priors$vdf
  return(list(Ef1=Ef1,Vf1=Vf1))
 }
```

D.3 Reified decision-dependent forecasting

Here we present some of the code used to generate the example in Chapter 5. Much of the code used to generate this example was the same as the code above. We do not include these functions here. The code in this section uses C to refer to H^1. Firstly we have the functions that generate and adjust our beliefs about β^*

```
BStars <- function(ERhoPrime, ERhoStar, Betas, VRhoPrime, VRhoStar){
  EBetaStar <- ERhoStar*ERhoPrime*Betas
  VBetaStar <- outer(Betas,Betas)*(VRhoStar*(VRhoPrime+(ERhoPrime^2))+
              (ERhoStar^2)*VRhoPrime)
  return(list(EBetaStar=EBetaStar,VBetaStar=VBetaStar))
}

AdjBStars <- function(C,ERhoStar,ERhoPrime,VRhoStar,VRhoPrime,Betas,VarOmega){
  AEBetaStar <- ERhoStar*C*Betas
  AVBetaStar <- outer(Betas,Betas)*(VRhoStar*(VRhoPrime+(ERhoPrime^2))+
                ((ERhoStar^2)*VarOmega))
  return(list(EBetaStar=AEBetaStar,VBetaStar=AVBetaStar))
```

```
}
```

The object *ReifiedModelStuff* is a list containing all of our beliefs regarding the emulators for the different models and other things required in order to produce forecasts. This is the equivalent of the list *ComputerModelStuff* in the code for the other example. The following two functions compute our beliefs about y having adjusted β^* by H^1, and our joint beliefs about y and H^1, respectively.

```
YmomentsGivenC <- function(theta,EBetaStar,VBetaStar,ReifiedModelStuff){
  rms <- ReifiedModelStuff
  reqInts <- RequiredIntegrals(theta=theta,Theta=rms$Theta,CLdec=rms$CLdec,
          DecisionVector=rms$DecisionVector,Rdec=rms$Rdec,V=rms$V,
          SIGu=rms$SIGu,hxInt=rms$hxInt,hxhxtInt=rms$hxhxtInt,rxInt=rms$rxInt,
          rxrxtInt=rms$rxrxtInt,hxrxtInt=rms$hxrxtInt)
  ERegress <- ExpectRegression(EBetaStar,gxInt=reqInts$gxInt)
  EResid <- ExpectResid(Q=rms$Q,HalfUp=rms$HalfUp,covuUInt=reqInts$covuUInt,
         InvVDat=rms$InvVDat,FminusEF=rms$FminusEF)
  Ey <- ERegress + EResid
  VBgInt <- VarBg(VBetaStar=VBetaStar,gxgxTInt=reqInts$gxgxTInt)
  EbgEbg <- IntEbgEbg(gxgxTInt=reqInts$gxgxTInt, EBetaStar=EBetaStar)
  AVux <- IntAdjux(Q=rms$Q,V=rms$V,SIGu=rms$SIGu,
       covuUcovUuInt=reqInts$covuUcovUuInt,InvVDat=rms$InvVDat)+
       diag((rms$SIGDeltaPrime)^2)+diag((rms$SIGDeltaStar)^2)
  AEuxAEux <- IntEuxEux(covuUcovUuInt=reqInts$covuUcovUuInt,HalfUp=rms$HalfUp,
           Q=rms$Q,InvVDat=rms$InvVDat,FminusEF=rms$FminusEF)
  muSquared <- outer(Ey,Ey)
  EbgEux <- IntEbgEux(covugIntegral=reqInts$covugIntegral,EBetaStar=EBetaStar,
         HalfUp=rms$HalfUp,Q=rms$Q,InvVDat=rms$InvVDat,FminusEF=rms$FminusEF)
  Vy <- VBgInt + AVux + EbgEbg + AEuxAEux + EbgEux + t(EbgEux)
        + rms$SIGdisStar - muSquared
  return(list(Ey=Ey,Vy=Vy,gxInt=reqInts$gxInt))
}

YJointwithC <- function(theta,EBetaStar,VBetaStar,ReifiedModelStuff,ERhoPrime,
                        ERhoStar,VRhoPrime,VarOmega,Betas){
  #We havn't adjusted B* by C here.
  rms <- ReifiedModelStuff
  Ymoms <- YmomentsGivenC(theta=theta,EBetaStar=EBetaStar,VBetaStar=VBetaStar,
```

```
        ReifiedModelStuff=rms)
  EyC <- c(Ymoms$Ey,ERhoPrime)
  VC <- VRhoPrime-VarOmega
  m <- length(Ymoms$gxInt)
  temp <- VC*ERhoStar*Betas
  covyC <- tensor(temp,Ymoms$gxInt,2,1)
  VyC <- matrix(0,nrow=length(EyC),ncol=length(EyC))
  n <- length(EyC)
  VyC[1:(n-1),1:(n-1)] <- Ymoms$Vy
  VyC[n,n] <- VC
  VyC[1:(n-1),n] <- covyC
  VyC[n,1:(n-1)] <- covyC
  return(list(EyC=EyC,VyC=VyC))
}
```

We compute the reified decision-dependent forecast or decision-dependent observation forecast as required. The code below is for the example only and deals with scalar observations at each time point. It would need generalizing in order to handle vectors of observations at each time point.

```
ReifiedForecast <- function(Expectation,Variance,Zh,Zt=NULL,
                            is.intervention=FALSE,SIGe){
  #Expectation is E[y] or E[y,C] depending on is.intervention
  #Variance is Var[y] or Var[y,C] depending on is.intervention.
  if(is.intervention){
    EztC <- Expectation[c(2,4)]
    VztC <- Variance[c(2,4),c(2,4)]
    VztC <- VztC + rbind(c(SIGe^2,0),c(0,0))
    Ezh <- Expectation[1]
    Vzh <- Variance[1,1] + SIGe^2
    covztCzh <- Variance[c(2,4),1]
    diff <- Zh - Ezh
    Qz <- chol(Vzh)
    x <- backsolve(Qz,diff,transpose=TRUE)
    y <- backsolve(Qz,t(covztCzh),transpose=TRUE)
    addition <- crossprod(y,x)
    dim(addition) <- dim(EztC)
    ExpectFor <- EztC + addition
```

```
    VarFor <- VztC - crossprod(y)
    return(list(Expectation=ExpectFor,Variance=VarFor))
  }
  else{
    Ezhzt <- Expectation[c(1,2)]
    Eyhytyf <- Expectation[1:3]
    data <- c(Zh,Zt)
    diff <- data-Ezhzt
    Vzhzt <- Variance[c(1,2),c(1,2)] + rbind(c(SIGe^2,0),c(0,SIGe^2))
    Covyhytyf.zhzt <- Variance[1:3,1:2]
    Qz <- chol(Vzhzt)
    x <- backsolve(Qz,diff,transpose=TRUE)
    y <- backsolve(Qz,t(Covyhytyf.zhzt),transpose=TRUE)
    dim(Eyhytyf) <- c(3,1)
    ExpectFor <- Eyhytyf + crossprod(y,x)
    VarFor <- Variance - crossprod(y)
    return(list(Expectation=ExpectFor,Variance=VarFor))
  }
}

ForecastY <- function(theta, Vw, C, Zt, ReifiedModelStuff, VRhoPrime, VRhoStar){
  rms <- ReifiedModelStuff
  VarOmega=Vw
  if(!((VRhoStar-Vw)>0))
    ABStar <- BStars(ERhoPrime=rms$ERhoPrime,ERhoStar=rms$ERhoStar,
           Betas=rms$Betas,VRhoPrime=VRhoPrime,VRhoStar=VRhoStar)
  else
    ABStar <- AdjBStars(C=C,ERhoStar=rms$ERhoStar,ERhoPrime=rms$ERhoPrime,
           VRhoPrime=VRhoPrime,VRhoStar=VRhoStar,Betas=rms$Betas,
           VarOmega=VarOmega)
  Ymoms <- YmomentsGivenC(theta=theta,EBetaStar=ABStar$EBetaStar,
        VBetaStar=ABStar$VBetaStar,ReifiedModelStuff=rms)
  ReifiedForecast(Expectation=Ymoms$Ey,Variance=Ymoms$Vy,Zh=rms$Zh,Zt=Zt,
                  SIGe=rms$SIGe)
}

ForecastZtC <- function(theta0,Vw,ReifiedModelStuff,VRhoPrime,VRhoStar){
```

```
  rms <- ReifiedModelStuff
  VarOmega <- Vw
  BStar <- BStars(ERhoPrime=rms$ERhoPrime,ERhoStar=rms$ERhoStar,Betas=rms$Betas,
        VRhoPrime=VRhoPrime,VRhoStar=VRhoStar)
  YCmoms <- YJointwithC(theta=c(theta0,0),EBetaStar=BStar$EBetaStar,
         VBetaStar=BStar$VBetaStar,ReifiedModelStuff=rms,ERhoPrime=rms$ERhoPrime,
         ERhoStar=rms$ERhoStar,VRhoPrime=VRhoPrime,VarOmega=VarOmega,
         Betas=rms$Betas)
  ReifiedForecast(Expectation=YCmoms$EyC,Variance=YCmoms$VyC,Zh=rms$Zh,
                  is.intervention=TRUE,SIGe=rms$SIGe)
}
```

Below we present the code used for generating samples from the expected losses $A(\theta^1, z^1, H^1)$, $B^1_{\lambda^1}(\theta_{t_0}, z^1, H^1)$, and $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0}, H^1)$. Certain functions, such as the function *t2Strategy* are the same in spirit to their counterparts above, but have more inputs. We do not include these, nor do we include the emulator functions *RAFine*, *RBFine* and *RCFine* for the same reason.

```
ASampleFun <- function(theta,Vw,C,Zt,ReifiedModelStuff,VRhoPrime,VRhoStar,
                       numSamples,diceParams,is.full=FALSE,nfSamples=NULL){
  Ymoms <- ForecastY(theta=theta,Vw=Vw,C=C,Zt=Zt,VRhoPrime=VRhoPrime,
        ReifiedModelStuff=ReifiedModelStuff,VRhoStar=VRhoStar)
  Csamples <- MultiLogSamples(numSamples,Ymoms$Expectation,Ymoms$Variance)
  CLosses <- sapply(1:numSamples,function(i) thisreifiedDICE(yh=Csamples[i,1],
          yt=Csamples[i,2],yf=Csamples[i,3],theta=theta,rho0=diceParams$rho0,
          gp=diceParams$gp,b10=diceParams$b10,gb0=diceParams$gb0,
          futureRate=diceParams$futureRate,Vw=Vw,VRhoPrime=VRhoPrime))
  ECloss <- sum(CLosses)/numSamples
  if(is.full){
    Fsamples <- MultiLogSamples(nfSamples,Ymoms$Expectation,Ymoms$Variance)
    FLosses <- sapply(1:nfSamples,function(i) thisreifiedDICE(yh=Fsamples[i,1],
            yt=Fsamples[i,2],yf=Fsamples[i,3],theta=theta,rho0=diceParams$rho0,
            gp=diceParams$gp,b10=diceParams$b10,gb0=diceParams$gb0,
            futureRate=diceParams$futureRate,Vw=Vw,VRhoPrime=VRhoPrime))
    return(cbind(ECloss,sum(FLosses)/nfSamples))
  }
  else
    return(ECloss)
```

```
}

ASamples <- function(Adesign,ReifiedModelStuff,VRhoPrime,VRhoStar,numSamples,
                  nfSamples=NULL,is.full=FALSE,diceParams){
   if(is.full & is.null(nfSamples))
     stop("Require a number for nfSamples or that is.full==FALSE")
   if(is.full){
     Losses <- sapply(1:length(Adesign[,1]),function(i) ASampleFun(
            theta=c(Adesign$t1[i],Adesign$t2[i]),Vw=Adesign$Vw[i],C=Adesign$C[i],
            Zt=Adesign$zt[i],ReifiedModelStuff=ReifiedModelStuff,
            VRhoPrime=VRhoPrime,VRhoStar=VRhoStar,numSamples=numSamples,
            is.full=TRUE,nfSamples=nfSamples,diceParams=diceParams))
     return(cbind(Adesign,ECloss=Losses[1,],EFloss=Losses[2,]))
   }
   else{
     Eloss <- sapply(1:length(Adesign[,1]),function(i) ASampleFun(
           theta=c(Adesign$t1[i],Adesign$t2[i]),Vw=Adesign$Vw[i],C=Adesign$C[i],
           Zt=Adesign$zt[i],ReifiedModelStuff=ReifiedModelStuff,
           VRhoPrime=VRhoPrime,VRhoStar=VRhoStar,numSamples=numSamples,
           diceParams=diceParams))
     return(cbind(Adesign,Eloss))
   }
 }

BSampleFun <- function(theta1,Vw,C,Zt,ReifiedModelStuff,VRhoPrime,VRhoStar,
           numSamples,diceParams,is.full=FALSE,nfSamples=NULL,AEmulator){
  t2 <- t2Strategy(t1=theta1,Vw=Vw,C=C,zt=Zt,AEmulator=AEmulator)
  theta=c(theta1,t2)
  Ymoms <- ForecastY(theta=c(theta1,t2),Vw=Vw,C=C,Zt=Zt,
        ReifiedModelStuff=ReifiedModelStuff,VRhoPrime=VRhoPrime,
        VRhoStar=VRhoStar)
  Csamples <- MultiLogSamples(numSamples,Ymoms$Expectation,Ymoms$Variance)
  CLosses <- sapply(1:numSamples,function(i) thisreifiedDICE(yh=Csamples[i,1],
          yt=Csamples[i,2],yf=Csamples[i,3],theta=theta,Vw=Vw,VRhoPrime=VRhoPrime,
          rho0=diceParams$rho0,gp=diceParams$gp,b10=diceParams$b10,
          gb0=diceParams$gb0,futureRate=diceParams$futureRate))
  ECloss <- sum(CLosses)/numSamples
```

```
  if(is.full){
    Fsamples <- MultiLogSamples(nfSamples,Ymoms$Expectation,Ymoms$Variance)
    FLosses <- sapply(1:nfSamples,function(i) thisreifiedDICE(yh=
            Fsamples[i,1],yt=Fsamples[i,2],yf=Fsamples[i,3],theta=theta,
            Vw=Vw,VRhoPrime=VRhoPrime,rho0=diceParams$rho0,gp=diceParams$gp,
            b10=diceParams$b10,gb0=diceParams$gb0,
            futureRate=diceParams$futureRate))
    return(cbind(ECloss,sum(FLosses)/nfSamples))
  }
  else
    return(ECloss)
}

BSamples <- function(Bdesign,ReifiedModelStuff,VRhoPrime,VRhoStar,numSamples,
            nfSamples=NULL,is.full=FALSE,diceParams,AEmulator){
   if(is.full & is.null(nfSamples))
     stop("Require a number for nfSamples or that is.full==FALSE")
   if(is.full){
     Losses <- sapply(1:length(Bdesign[,1]),function(i) BSampleFun(theta1=
            Bdesign$t1[i],Vw=Bdesign$Vw[i],C=Bdesign$C[i],Zt=Bdesign$zt[i],
            ReifiedModelStuff=ReifiedModelStuff,VRhoPrime=VRhoPrime,
            numSamples=numSamples,nfSamples=nfSamples,VRhoStar=VRhoStar,
            diceParams=diceParams,AEmulator=AEmulator,is.full=TRUE))
     return(cbind(Bdesign,ECloss=Losses[1,],EFloss=Losses[2,]))
   }
   else{
     Eloss <- sapply(1:length(Bdesign[,1]),function(i) BSampleFun(theta1=
           Bdesign$t1[i],Vw=Bdesign$Vw[i],C=Bdesign$C[i],Zt=Bdesign$zt[i],
           ReifiedModelStuff=ReifiedModelStuff,VRhoPrime=VRhoPrime,
           VRhoStar=VRhoStar,numSamples=numSamples,diceParams=diceParams,
           AEmulator=AEmulator))
     return(cbind(Bdesign,Eloss))
   }
 }

CSampleFun <- function(theta1,Vw,ReifiedModelStuff,VRhoPrime,VRhoStar,numSamples,
              BEmulator,is.full=FALSE,nfSamples=NULL,Cmax){
```

```
rms <- ReifiedModelStuff
ZtCMoms <- ForecastZtC(theta0=theta1,Vw=Vw,ReifiedModelStuff=rms,
        VRhoPrime=VRhoPrime,VRhoStar=VRhoStar)
if(!((VRhoPrime-Vw)>0)){
  Csamples <- UniLogSamples(numSamples,ZtCMoms$Expectation[1],
           ZtCMoms$Variance[1,1])
  CLosses <- sapply(1:numSamples,function(i) RBFine(t1=theta1,Vw=Vw,C=1,
          zt=Csamples[i],Coarse=BEmulator$Coarse,BuiltFine=BEmulator$BuiltFine,
          Priors=BEmulator$Priors,D=BEmulator$D)$Ef1)
}
else{
  Csamples <- MultiLogSamples(numSamples,ZtCMoms$Expectation,ZtCMoms$Variance)
  RejectAndComplete <- function(samples,numRequired,maxSam){
    Csams <- samples[,2]
    numWrong <- sum(Csams>maxSam)
    if((numRequired-numWrong)<numRequired){
      samples <- samples[samples[,2]<=maxSam,]
      newSams <- MultiLogSamples(numWrong,ZtCMoms$Expectation,ZtCMoms$Variance)
      newsamples <- rbind(samples,newSams)
      samples <- RejectAndComplete(newsamples,numRequired,maxSam)
    }
    else{
      return(samples)
    }
  }
  Csamples <- RejectAndComplete(Csamples,numSamples,maxSam=Cmax)
  CLosses <- sapply(1:numSamples,function(i) RBFine(t1=theta1,
          Vw=Vw,C=Csamples[i,2],zt=Csamples[i,1],Coarse=BEmulator$Coarse,
          BuiltFine=BEmulator$BuiltFine,
          Priors=BEmulator$Priors,D=BEmulator$D)$Ef1)
}
ECloss <- sum(CLosses)/numSamples
if(is.full){
  if(!((VRhoPrime-Vw)>0)){
    Fsamples <- UniLogSamples(nfSamples,ZtCMoms$Expectation[1],
             ZtCMoms$Variance[1,1])
    FLosses <- sapply(1:nfSamples,function(i) RBFine(t1=theta1,Vw=Vw,C=1,
```

```
            zt=Fsamples[i],Coarse=BEmulator$Coarse,
            BuiltFine=BEmulator$BuiltFine,
            Priors=BEmulator$Priors,D=BEmulator$D)$Ef1)
    }
    else{
      Fsamples <- MultiLogSamples(nfSamples,ZtCMoms$Expectation,ZtCMoms$Variance)
      FLosses <- sapply(1:nfSamples,function(i) RBFine(t1=theta1,Vw=Vw,
            C=Fsamples[i,2],zt=Fsamples[i,1],Coarse=BEmulator$Coarse,
            D=BEmulator$D,BuiltFine=BEmulator$BuiltFine,
            Priors=BEmulator$Priors)$Ef1)
    }
    return(cbind(ECloss,sum(FLosses)/nfSamples))
  }
  else{
    return(ECloss)
  }
}

CSamples <- function(Cdesign,ReifiedModelStuff,VRhoPrime,VRhoStar,is.full=FALSE,
           numSamples,nfSamples=NULL,BEmulator,Cmax){
  if(is.full & is.null(nfSamples))
    stop("Require a number for nfSamples or that is.full==FALSE")
  if(is.full){
    Losses <- sapply(1:length(Cdesign[,1]),function(i) CSampleFun(theta1=
           Cdesign$t1[i],Vw=Cdesign$Vw[i],ReifiedModelStuff=ReifiedModelStuff,
           VRhoPrime=VRhoPrime,VRhoStar=VRhoStar,numSamples=numSamples,
           is.full=TRUE,nfSamples=nfSamples,Cmax=Cmax,BEmulator=BEmulator))
    return(cbind(Cdesign,ECloss=Losses[1,],EFloss=Losses[2,]))
  }
  else{
    Eloss <- sapply(1:length(Cdesign[,1]),function(i) CSampleFun(theta1=
          Cdesign$t1[i],Vw=Cdesign$Vw[i],ReifiedModelStuff=ReifiedModelStuff,
          VRhoPrime=VRhoPrime,VRhoStar=VRhoStar,numSamples=numSamples,
          Cmax=Cmax,BEmulator=BEmulator))
    return(cbind(Cdesign,Eloss))
  }
}
```

Appendix E

Miscellaneous details of built emulators

E.1 The Emulator for C-GOLDSTEIN

We present here a table of adjusted expectations for the regression coefficients in our emulator for C-GOLDSTEIN. We now compute the adjustment discrepancy,

	k=1	k=2	k=3
$E_F[\beta_{k1}]$	0.454125	0.8351274	0.9356258
$E_F[\beta_{k2}]$	0.04512565	0.16365801	0.28487865
$E_F[\beta_{k3}]$	-0.01630596	-0.10221798	-0.20931688
$E_F[\beta_{k4}]$	0.01506409	-0.04864731	-0.12295281
$E_F[\beta_{k5}]$	0	0.3309102	0.4178705
$E_F[\beta_{k6}]$	0	0.1201084	0.1415842
$E_F[\beta_{k7}]$	0	0	1.050493
$E_F[\beta_{k8}]$	0	0	0.1070709

Table E.1: Adjusted expectations for the regression coefficients in our emulator for C-Goldstein.

$Dis_F(\cdot)$, as defined by equation (4.8), for each of the regression coefficients. High values of the adjustment discrepancy often indicate surprising adjusted values and

	k=1	k=2	k=3
$Dis_F(\beta_{k1})$	0.02867263	0.005783111	3.2086682
$Dis_F(\beta_{k2})$	18.75178	29.87757	21.40507
$Dis_F(\beta_{k3})$	17.08221	36.18681	22.88085
$Dis_F(\beta_{k4})$	0.6638616	0.5088078	4.041319
$Dis_F(\beta_{k5})$	0	7.463643	10.40727
$Dis_F(\beta_{k6})$	0	0.0655515	0.1554626
$Dis_F(\beta_{k7})$	0	0	4.646366
$Dis_F(\beta_{k8})$	0	0	8.609593

Table E.2: Adjustment discrepancy for the regression coefficients in our emulator for C-Goldstein.

can be a symptom of a problem with our prior specification. There are a number of discrepancies in table E.2 that are particularly large. A more detailed look at the calculation of these numbers revealed that the resolved variances were mostly of order 10^{-5}, and that these small numbers were the cause of the large discrepancies we were observing. This suggests that we may have been over confident in our prior specification of the regression coefficients. We also noticed that our update resolved only a very small proportion of the variance we had on the coefficients. This perhaps suggests that we have not made enough runs on our simulator. Further investigation showed that even if we had resolved all of our prior variance, these discrepancies would be unusually high, indicating an over confidence in our prior specification.

In order to explore whether or not this over confidence in our prior specification needs to be addressed, we explored the impact that increasing the variances would have on our analysis. We multiplied all variances on the regression coefficients by ten and calculated the adjustment discrepancies under this new prior specification. These alternative discrepancies were small enough not to flag up anything unusual in our specification. As the variance of the regression coefficients was already small, the real question was how much would increasing these variances by a factor of ten change the variance of our forecasts for y.

We computed forecasts using our original emulator for C-GOLDSTEIN and using

an emulator built having multiplied our variance on the regression coefficients by ten. The forecast variance is a matrix of numbers of order 10^{-1} under either variance specification. We computed the difference between the forecast variances for a wide range of θ and found no value larger than 5×10^{-3} in any difference matrix. A typical value of this difference matrix is shown below:

$$\begin{pmatrix} 0.0001351782 & 0.00003117255 & 0.0003007227 \\ 0.00003117255 & 0.000002726135 & 0.00007550926 \\ 0.0003.007227 & 0.00007550926 & 0.004739902 \end{pmatrix}.$$

We concluded that our over confidence in our specification of the regression coefficients had a negligible effect on our analysis and therefore decided not to revisit our prior specification. A more careful analysis for a real-world decision problem may go further than we have here and either decide to revisit the prior specification anyway, or consider further Bayes Linear diagnostics and look at how slight changes in our forecast variance propagate through to our expected loss surface. It was our feeling, however, that other simplifications we have made, such as the choice of a log-normal distribution for future climate, have such an impact on our analysis that these very small changes in variance are not worth worrying about.

E.2 Sequential Emulation in Chapter 4

E.2.1 The emulator of section 4.6.3

Here we present the linear models we looked at when deciding which regression surface to use when fitting the emulator for $A(\theta^1, z^1)$. We also present some of the linear models we looked at as part of diagnostic checks. When performing the Sequential Emulation in R, we use the notation $t1 = \theta_{t_0}$, $t2 = \theta_{t_1}$, $zt = z_{t_1}$. This is for programming convenience only, however it does mean that the output we present here has a different notation to the rest of the thesis.

Here is the saturated linear model of order 2 fitted to all of the data.

```
 Call:
lm(formula = Eloss ~ I(zt) + I(t2) + I(t1) + I(zt^2) + I(t2 *
```

```
    zt) + I(t2^2) + I(t1 * zt) + I(t1 * t2) + I(t1^2), data = fasta)


Residuals:
     Min       1Q   Median       3Q      Max
-0.39749 -0.05878 -0.01022  0.05639  0.58463


Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.0948712  0.0111765  -8.488  < 2e-16 ***
I(zt)       -0.0558147  0.0060116  -9.285  < 2e-16 ***
I(t2)       -0.0068202  0.0113707  -0.600   0.5488
I(t1)       -0.0649673  0.0112548  -5.772 1.04e-08 ***
I(zt^2)      0.1558204  0.0007547 206.455  < 2e-16 ***
I(t2 * zt)   0.0281378  0.0028026  10.040  < 2e-16 ***
I(t2^2)      0.5381425  0.0119672  44.968  < 2e-16 ***
I(t1 * zt)   0.0258177  0.0027630   9.344  < 2e-16 ***
I(t1 * t2)   0.0659274  0.0107951   6.107 1.45e-09 ***
I(t1^2)      0.0287022  0.0119402   2.404   0.0164 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1


Residual standard error: 0.1125 on 990 degrees of freedom
Multiple R-squared: 0.9982,Adjusted R-squared: 0.9982
F-statistic: 6.176e+04 on 9 and 990 DF,  p-value: < 2.2e-16
```

We expected that most of the signal was due to z_{t_1}, having seen figure 4.17. The coefficients indicate that this is the case, with the largest contribution coming from the coefficient of $z_{t_1}^2$. However, there is an important signal due to $\theta_{t_1}^2$ indicated by the relatively large value of that coefficient in the above model.

Here is the saturated model of order 3 fitted to all of the data.

```
 Call:
lm(formula = Eloss ~ I(zt) + I(t2) + I(t1) + I(zt^2) + I(t2 *
```

```
    zt) + I(t2^2) + I(t1 * zt) + I(t1 * t2) + I(t1^2) + I(zt^3) +
    I(t2 * zt^2) + I(t2^2 * zt) + I(t2^3) + I(t1 * zt^2) + I(t1 *
    t2 * zt) + I(t1 * t2^2) + I(t1^2 * zt) + I(t1^2 * t2) + I(t1^3),
    data = fasta)

Residuals:
       Min         1Q     Median         3Q        Max
-0.1784679 -0.0216735 -0.0009968  0.0199730  0.1842016

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      0.1344386  0.0054203  24.803  < 2e-16 ***
I(zt)           -0.2042197  0.0048498 -42.109  < 2e-16 ***
I(t2)            0.0106723  0.0075956   1.405 0.160316
I(t1)           -0.1025650  0.0074901 -13.693  < 2e-16 ***
I(zt^2)          0.1881912  0.0014642 128.524  < 2e-16 ***
I(t2 * zt)       0.0343961  0.0034604   9.940  < 2e-16 ***
I(t2^2)          0.0406021  0.0072355   5.612 2.61e-08 ***
I(t1 * zt)       0.0359073  0.0034159  10.512  < 2e-16 ***
I(t1 * t2)      -0.0150607  0.0066053  -2.280 0.022816 *
I(t1^2)          0.0247299  0.0071865   3.441 0.000604 ***
I(zt^3)         -0.0027570  0.0001249 -22.072  < 2e-16 ***
I(t2 * zt^2)    -0.0004794  0.0004348  -1.102 0.270554
I(t2^2 * zt)     0.1475210  0.0017815  82.807  < 2e-16 ***
I(t2^3)         -0.0756858  0.0077718  -9.739  < 2e-16 ***
I(t1 * zt^2)    -0.0011894  0.0004306  -2.762 0.005850 **
I(t1 * t2 * zt)  0.0278729  0.0016137  17.273  < 2e-16 ***
I(t1 * t2^2)     0.0710246  0.0068513  10.367  < 2e-16 ***
I(t1^2 * zt)     0.0006436  0.0017801   0.362 0.717770
I(t1^2 * t2)     0.0308151  0.0068677   4.487 8.08e-06 ***
I(t1^3)         -0.0149046  0.0077445  -1.925 0.054575 .
```

```
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.03691 on 980 degrees of freedom
Multiple R-squared: 0.9998,Adjusted R-squared: 0.9998
F-statistic: 2.721e+05 on 19 and 980 DF,  p-value: < 2.2e-16
```

The quadratic term in z_{t_1} here is still important as we might have expected. What we also notice is that the interaction between $\theta_{t_1}^2$ and z_{t_1} is important. The signal that we saw in the residual plots of figure 4.18 can be attributed to this relationship and by fitting these extra terms we have reduced the residual variance by roughly two thirds. We also notice that the main effects in θ_{t_0} appear to be linear.

Here is the saturated model of order 2 fitted to the subset of the data with $z_{t_1} < 3$.

```
 Call:
lm(formula = Eloss ~ I(zt) + I(t2) + I(t1) + I(zt^2) + I(t2 *
    zt) + I(t2^2) + I(t1 * zt) + I(t1 * t2) + I(t1^2), data = fasta1)

Residuals:
      Min        1Q    Median        3Q       Max
-0.170368 -0.033395 -0.007918  0.028277  0.205347

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.045680   0.007925   5.764 1.47e-08 ***
I(zt)       -0.108843   0.010730 -10.144  < 2e-16 ***
I(t2)       -0.013281   0.008224  -1.615  0.10698
I(t1)       -0.068646   0.008009  -8.571  < 2e-16 ***
I(zt^2)      0.165953   0.003459  47.973  < 2e-16 ***
I(t2 * zt)   0.031658   0.004713   6.716 5.30e-11 ***
I(t2^2)      0.255036   0.007865  32.426  < 2e-16 ***
I(t1 * zt)   0.021441   0.004677   4.584 5.82e-06 ***
```

```
I(t1 * t2)   0.023180   0.007208   3.216  0.00139 **
I(t1^2)      0.018100   0.007858   2.303  0.02169 *
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.05114 on 478 degrees of freedom
Multiple R-squared: 0.9814,Adjusted R-squared: 0.9811
F-statistic:  2806 on 9 and 478 DF,  p-value: < 2.2e-16
```

Here is the saturated model of order 3 fitted to the subset of the data with $z_{t_1} < 3$.

```
 Call:
lm(formula = Eloss ~ I(zt) + I(t2) + I(t1) + I(zt^2) + I(t2 *
    zt) + I(t2^2) + I(t1 * zt) + I(t1 * t2) + I(t1^2) + I(zt^3) +
    I(t2 * zt^2) + I(t2^2 * zt) + I(t2^3) + I(t1 * zt^2) + I(t1 *
    t2 * zt) + I(t1 * t2^2) + I(t1^2 * zt) + I(t1^2 * t2) + I(t1^3),
    data = fasta1)

Residuals:
      Min        1Q    Median        3Q       Max
-0.067111 -0.017155 -0.001546  0.015925  0.081667

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      0.1259704  0.0058393  21.573  < 2e-16 ***
I(zt)           -0.1681915  0.0130753 -12.863  < 2e-16 ***
I(t2)           -0.0018350  0.0075664  -0.243 0.808483
I(t1)           -0.1168496  0.0076535 -15.267  < 2e-16 ***
I(zt^2)          0.1664698  0.0099405  16.747  < 2e-16 ***
I(t2 * zt)       0.0190264  0.0089794   2.119 0.034627 *
I(t2^2)          0.0193445  0.0077080   2.510 0.012422 *
I(t1 * zt)       0.0347162  0.0089152   3.894 0.000113 ***
```

```
I(t1 * t2)      -0.0067133  0.0070934  -0.946 0.344423
I(t1^2)          0.0261069  0.0073857   3.535 0.000449 ***
I(zt^3)          0.0005508  0.0021771   0.253 0.800397
I(t2 * zt^2)     0.0048160  0.0029122   1.654 0.098849 .
I(t2^2 * zt)     0.1599944  0.0044595  35.877  < 2e-16 ***
I(t2^3)         -0.0302164  0.0073767  -4.096 4.95e-05 ***
I(t1 * zt^2)    -0.0009645  0.0028918  -0.334 0.738878
I(t1 * t2 * zt)  0.0231088  0.0041050   5.629 3.12e-08 ***
I(t1 * t2^2)     0.0685646  0.0067767  10.118  < 2e-16 ***
I(t1^2 * zt)    -0.0015632  0.0043018  -0.363 0.716474
I(t1^2 * t2)     0.0057702  0.0066483   0.868 0.385879
I(t1^3)          0.0100555  0.0074849   1.343 0.179783
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.0243 on 468 degrees of freedom
Multiple R-squared: 0.9959,Adjusted R-squared: 0.9957
F-statistic:  5974 on 19 and 468 DF,  p-value: < 2.2e-16
```

We notice from this output that the same model terms are important even on the reduced space of z_{t_1}. We also notice that these coefficients are roughly the same size as with the model fitted to the full data. This is a good indication that the large R^2 values, which are an artifact of the very strong signal in z_{t_1} and the fact that we are forced to fit artificially large values of z_{t_1}, do not hide a different type of signal for low values of z_{t_1}.

E.2.2 The emulator of section 4.6.5

Here is the saturated linear model of order 2 fitted to all of the coarse evaluations we made of $B^1_{\lambda^1}(\theta_{t_0}, z^1)$.

```
 Call:
lm(formula = Eloss ~ I(zt) + I(t1) + I(zt^2) + I(t1 * zt) + I(t1^2),
```

```
    data = fastb)

Residuals:
      Min        1Q    Median        3Q       Max
-0.172964 -0.023741 -0.001372  0.024253  0.144069

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.094098   0.003793   24.81  < 2e-16 ***
I(zt)       -0.128482   0.002166  -59.33  < 2e-16 ***
I(t1)       -0.098721   0.004035  -24.46  < 2e-16 ***
I(zt^2)      0.172407   0.000272  633.89  < 2e-16 ***
I(t1 * zt)   0.019664   0.001002   19.63  < 2e-16 ***
I(t1^2)      0.022957   0.004299    5.34 1.15e-07 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.04047 on 994 degrees of freedom
Multiple R-squared: 0.9998,Adjusted R-squared: 0.9998
F-statistic: 9.484e+05 on 5 and 994 DF,  p-value: < 2.2e-16
```

We notice that the large effects are in z_{t_1} and that the influence of θ_{t_1} is minimal in this model. We expect this from the picture of the data shown in figure 4.21, and from our emulator for $A(\theta^1, z^1)$.

Here is the saturated linear model of order 3 fitted to all of the coarse evaluations that we made of $B^1_{\lambda^1}(\theta_{t_0}, z^1)$.

```
 Call:
lm(formula = Eloss ~ I(zt) + I(t1) + I(zt^2) + I(t1 * zt) + I(t1^2) +
    I(zt^3) + I(t1 * zt^2) + I(t1^2 * zt) + I(t1^3), data = fastb)

Residuals:
      Min        1Q    Median        3Q       Max
```

```
-0.142083 -0.019463 -0.001263  0.019408  0.130748


Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.1505061  0.0042888  35.093  < 2e-16 ***
I(zt)        -0.2182649  0.0042151 -51.781  < 2e-16 ***
I(t1)        -0.1021594  0.0062232 -16.416  < 2e-16 ***
I(zt^2)       0.2022714  0.0012777 158.310  < 2e-16 ***
I(t1 * zt)    0.0252363  0.0029647   8.512  < 2e-16 ***
I(t1^2)       0.0225635  0.0062893   3.588  0.00035 ***
I(zt^3)      -0.0025865  0.0001090 -23.722  < 2e-16 ***
I(t1 * zt^2) -0.0007666  0.0003753  -2.043  0.04135 *
I(t1^2 * zt) -0.0005389  0.0015504  -0.348  0.72822
I(t1^3)      -0.0033545  0.0067609  -0.496  0.61989
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1


Residual standard error: 0.0322 on 990 degrees of freedom
Multiple R-squared: 0.9999,Adjusted R-squared: 0.9999
F-statistic: 8.325e+05 on 9 and 990 DF,  p-value: < 2.2e-16
```

We notice that the cubic order terms hardly seem to have any impact and that the size of the coefficients on the quadratic terms is roughly the same. We also notice that the residual variation has hardly reduced even though we have fitted higher order terms.

Here is the saturated model of order 2 fitted to the subset of the data with $z_{t_1} < 3$.

```
 Call:
lm(formula = Eloss ~ I(zt) + I(t1) + I(zt^2) + I(t1 * zt) + I(t1^2),
    data = fastb1)


Residuals:
```

```
      Min        1Q    Median        3Q       Max
-0.054360 -0.016011 -0.001363  0.014159  0.097887


Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.134839   0.003428  39.331  < 2e-16 ***
I(zt)       -0.178118   0.004868 -36.590  < 2e-16 ***
I(t1)       -0.106030   0.003593 -29.507  < 2e-16 ***
I(zt^2)      0.182085   0.001567 116.173  < 2e-16 ***
I(t1 * zt)   0.024487   0.002097  11.679  < 2e-16 ***
I(t1^2)      0.020852   0.003562   5.853 8.91e-09 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1


Residual standard error: 0.0232 on 482 degrees of freedom
Multiple R-squared: 0.9955,Adjusted R-squared: 0.9955
F-statistic: 2.149e+04 on 5 and 482 DF,  p-value: < 2.2e-16
```

Having checked that our high R^2 is not merely a result of fitting over a large range of z_{t_1} in the same way we did for the previous emulator, we see that the coefficients do not greatly differ from those in the quadratic fit to all of the data. This gives us confidence that the fit does not look so good solely because we fit over a large range of z_{t_1} and that over the range of z_{t_1} that we will obtain most samples from, our emulator is still good.

E.2.3 The emulator of section 4.6.6

Here is the regression summary of the quadratic fit to the data presented in figure 4.24.

```
Call:
lm(formula = Eloss ~ I(t1) + I(t1^2), data = fastc)
```

```
Residuals:
       Min         1Q     Median         3Q        Max
-0.0198875 -0.0054529 -0.0004409  0.0047047  0.0285465

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.1551933  0.0003662  423.77   <2e-16 ***
I(t1)       -0.0227086  0.0004229  -53.70   <2e-16 ***
I(t1^2)      0.0532471  0.0008188   65.03   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.007721 on 997 degrees of freedom
Multiple R-squared: 0.8771,Adjusted R-squared: 0.8768
F-statistic:  3557 on 2 and 997 DF,  p-value: < 2.2e-16
```

Here is the regression summary of the cubic fit to the data presented in figure 4.24.

```
 Call:
lm(formula = Eloss ~ I(t1) + I(t1^2) + I(t1^3), data = fastc)

Residuals:
       Min         1Q     Median         3Q        Max
-0.0171705 -0.0043670 -0.0004053  0.0039921  0.0246317

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.1551932  0.0003013  515.08   <2e-16 ***
I(t1)       -0.0401153  0.0008697  -46.12   <2e-16 ***
I(t1^2)      0.0532494  0.0006737   79.05   <2e-16 ***
I(t1^3)      0.0290074  0.0013283   21.84   <2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1
```

```
Residual standard error: 0.006352 on 996 degrees of freedom
Multiple R-squared: 0.9169,Adjusted R-squared: 0.9166
F-statistic:  3662 on 3 and 996 DF,  p-value: < 2.2e-16
```

E.3 Policy Support in Chapter 4

E.3.1 More strategy plots

Figures E.1, E.2 and E.3 are further examples of the panels of strategy plots we presented in figure 4.29. We include these for interested readers.

E.3.2 Details of the Sequential Emulation on the pruned space

Here is the regression surface we fit to the new coarse data for $A(\theta^1, z^1)$

```
Call:
lm(formula = Eloss ~ t1 * t2 * zt + I(t1^2) + I(t2^2) + I(zt^2) +
    I(t1^3) + I(t2^3) + I(zt^3) + I(t1 * t2^2) + I(t1 * zt^2) +
    I(t2 * t1^2) + I(t2 * zt^2) + I(zt * t1^2) + I(zt * t2^2),
    data = pcsepa)

Residuals:
       Min         1Q     Median         3Q        Max
-0.1370837 -0.0190038 -0.0002835  0.0185715  0.1398779

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.505e-01  4.409e-03  34.125  < 2e-16 ***
t1           -1.331e-01  8.165e-03 -16.300  < 2e-16 ***
t2           -6.261e-03  9.359e-03  -0.669 0.503615
```

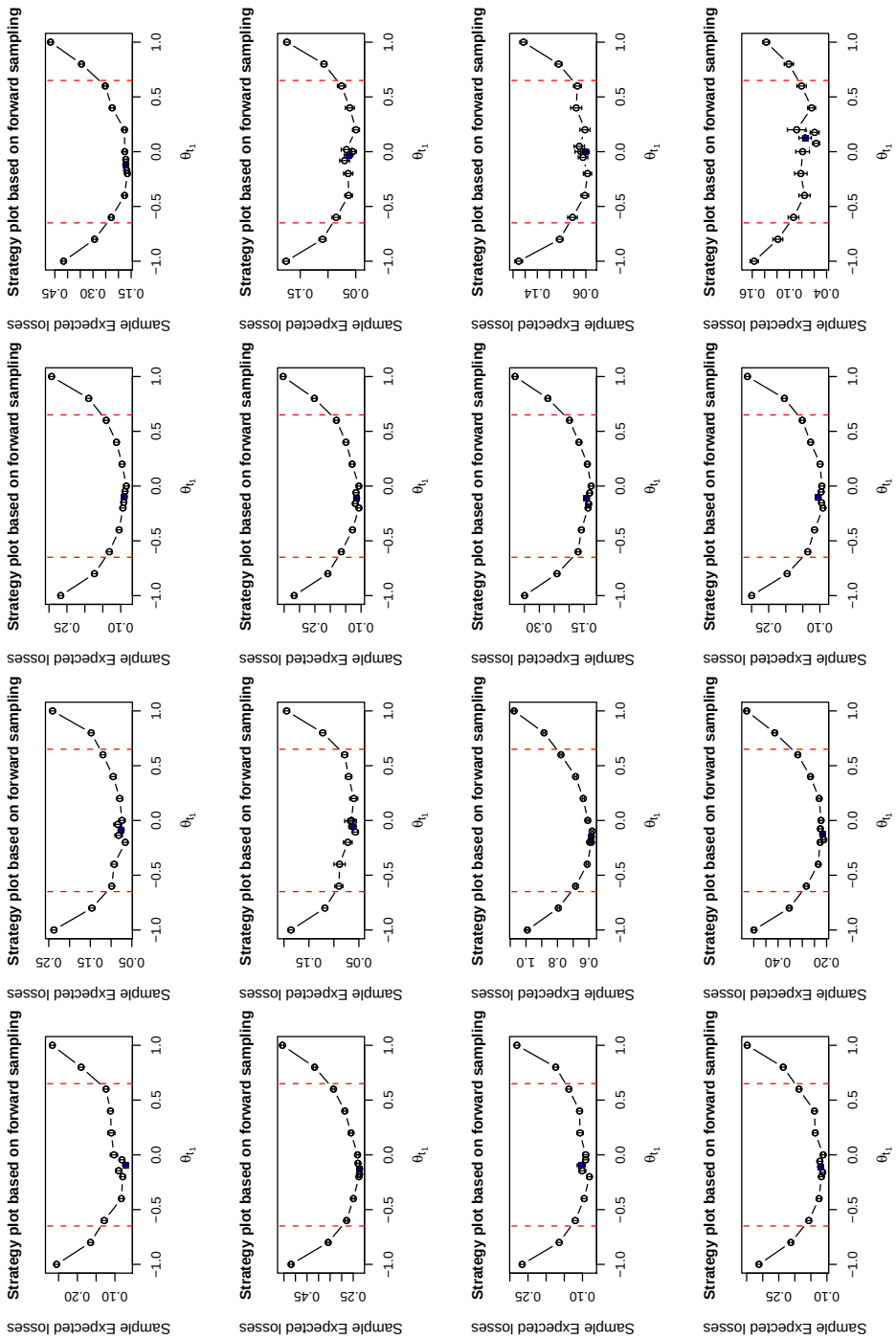

Figure E.1: Panel of strategy plots for $\theta_{t_0} = arg\min_{\theta_{t_0}} \{\tilde{E}\left[C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})\right]\}$. Each panel corresponds to one sampled z_{t_1}. Red dotted lines on each panel indicate $\theta_{t_1} = -0.65$ and $\theta_{t_1} = 0.65$.

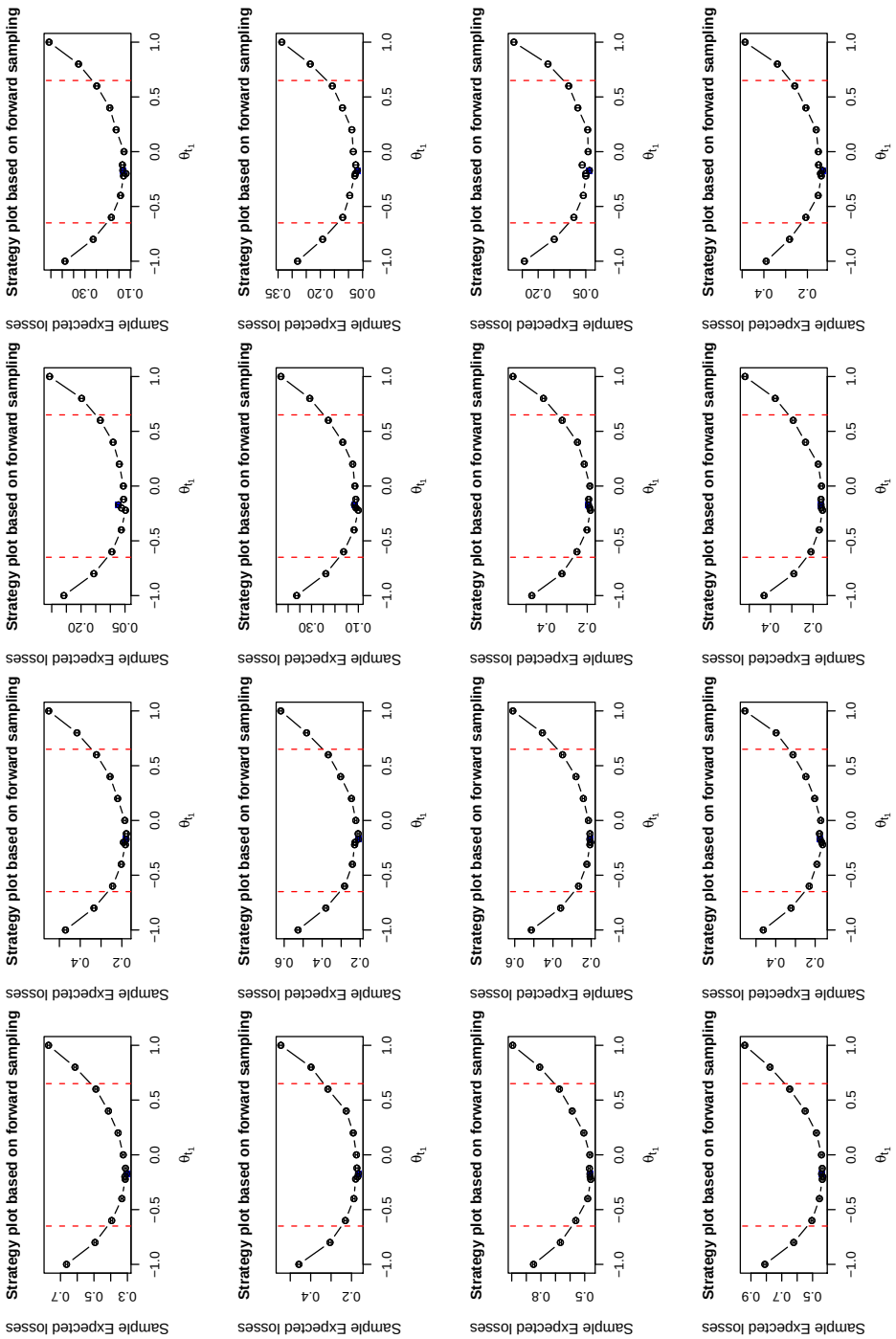

Figure E.2: Panel of strategy plots for $\theta_{t_0} = 0.95$. Each panel corresponds to one sampled z_{t_1}. Red dotted lines on each panel indicate $\theta_{t_1} = -0.65$ and $\theta_{t_1} = 0.65$.

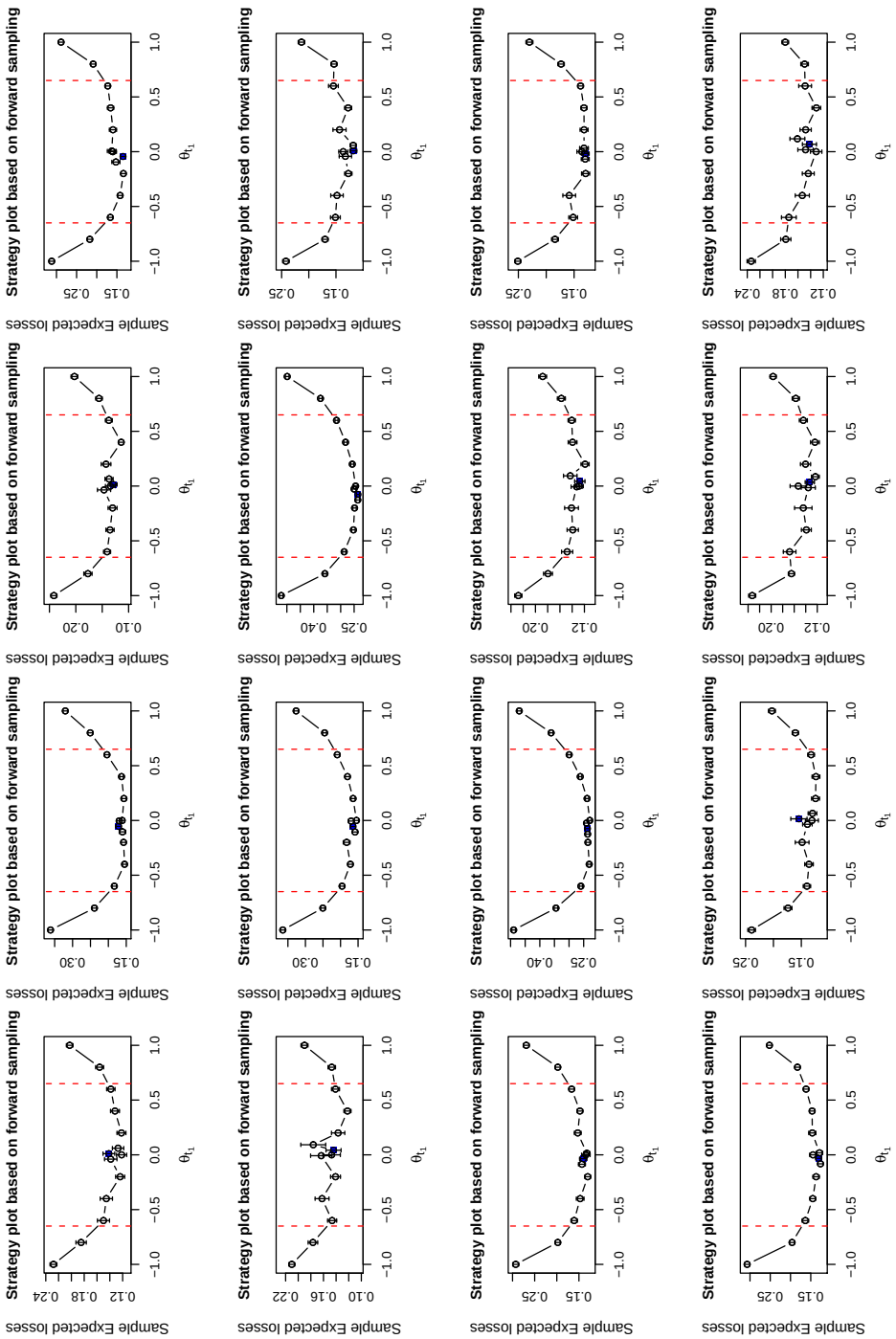

Figure E.3: Panel of strategy plots for $\theta_{t_0} = 0.4$. Each panel corresponds to one sampled z_{t_1}. Red dotted lines on each panel indicate $\theta_{t_1} = -0.65$ and $\theta_{t_1} = 0.65$.

```
zt           -2.108e-01  3.822e-03 -55.149  < 2e-16 ***
I(t1^2)       7.247e-02  1.295e-02   5.594 2.63e-08 ***
I(t2^2)      -3.672e-02  1.400e-02  -2.622 0.008832 **
I(zt^2)       1.927e-01  1.090e-03 176.840  < 2e-16 ***
I(t1^3)      -3.183e-02  1.271e-02  -2.504 0.012402 *
I(t2^3)      -1.081e-01  2.134e-02  -5.067 4.56e-07 ***
I(zt^3)      -2.681e-03  8.999e-05 -29.796  < 2e-16 ***
I(t1 * t2^2)  1.607e-01  1.651e-02   9.734  < 2e-16 ***
I(t1 * zt^2) -3.682e-04  4.318e-04  -0.853 0.393946
I(t2 * t1^2)  6.813e-02  1.373e-02   4.962 7.76e-07 ***
I(t2 * zt^2) -3.157e-04  5.121e-04  -0.616 0.537673
I(zt * t1^2) -7.874e-03  2.134e-03  -3.690 0.000233 ***
I(zt * t2^2)  1.465e-01  3.046e-03  48.105  < 2e-16 ***
t1:t2        -5.874e-02  1.219e-02  -4.819 1.59e-06 ***
t1:zt         3.702e-02  3.672e-03  10.081  < 2e-16 ***
t2:zt         4.309e-02  4.194e-03  10.276  < 2e-16 ***
t1:t2:zt      3.741e-02  2.389e-03  15.658  < 2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.03375 on 1489 degrees of freedom
Multiple R-squared: 0.9999,Adjusted R-squared: 0.9999
F-statistic: 5.604e+05 on 19 and 1489 DF,  p-value: < 2.2e-16
```

One thing to notice is the more substantial effect of θ_{t_0} in this model compared to the regression fit on the whole space. We now have a relatively large linear effect in θ_{t_0}, as well as an non negligible effect from the interaction of all three variables. One reason for this could be that our design on the pruned space sampled areas of the decision space that were important, but that had not been captured by our original designs. Another reason could be that, because our design concentrates the points in the pruned region, we are capturing smaller scale effects that our previous design did not capture. Here is the regression surface we fit to the new coarse data

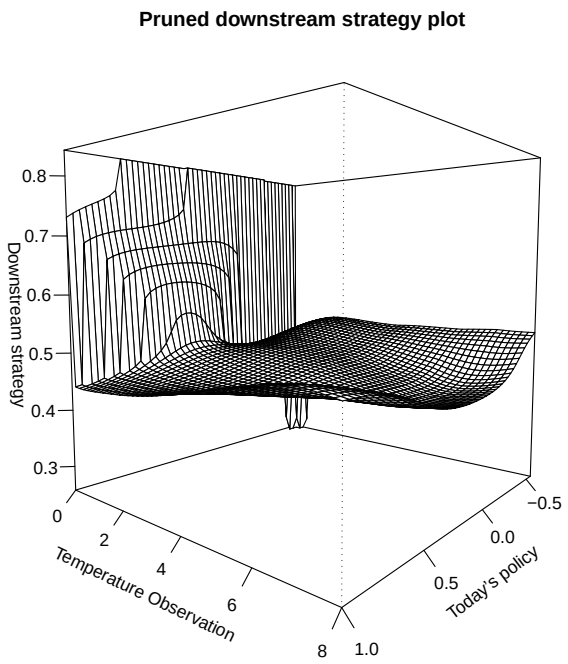

Figure E.4: Time t_1 strategy plot for θ_{t_1}, where the surface $\lambda_{t_1}(\theta_{t_0}, \theta^1)$, as calculated via our emulator for $A(\theta^1, z^1)$ on the pruned decision space, is plotted as a function of θ_{t_0} and z_{t_1}.

for $B^1_{\lambda^1}(\theta_{t_0}, z^1)$.

```
 Call:
lm(formula = Eloss ~ I(zt) + I(t1) + I(zt^2) + I(t1 * zt) + I(t1^2),
    data = fastb)


Residuals:
      Min        1Q    Median        3Q       Max
-0.201512 -0.028580 -0.003775  0.028411  0.143516


Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.1134677  0.0036176  31.365   <2e-16 ***
I(zt)       -0.1353137  0.0020907 -64.720   <2e-16 ***
```

```
I(t1)       -0.1020596  0.0058376 -17.483   <2e-16 ***
I(zt^2)      0.1732613  0.0002606 664.740   <2e-16 ***
I(t1 * zt)   0.0219657  0.0011982  18.332   <2e-16 ***
I(t1^2)     -0.0030731  0.0069434  -0.443    0.658
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.04276 on 1194 degrees of freedom
Multiple R-squared: 0.9998,Adjusted R-squared: 0.9998
F-statistic: 1.168e+06 on 5 and 1194 DF,  p-value: < 2.2e-16
```

Here is the regression surface we fit to the new coarse data for $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0})$.

```
 Call:
lm(formula = Eloss ~ I(t1) + I(t1^2) + I(t1^3), data = fastc)

Residuals:
       Min         1Q     Median         3Q        Max
-0.0154068 -0.0041001 -0.0003876  0.0035783  0.0267277

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.1598085  0.0003196 500.062  < 2e-16 ***
I(t1)       -0.0614532  0.0008400 -73.160  < 2e-16 ***
I(t1^2)      0.0813708  0.0020371  39.944  < 2e-16 ***
I(t1^3)      0.0206123  0.0026302   7.837 1.18e-14 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.005917 on 996 degrees of freedom
Multiple R-squared: 0.9071,Adjusted R-squared: 0.9068
F-statistic:  3241 on 3 and 996 DF,  p-value: < 2.2e-16
```

E.4 The Reified Sequential Emulation example

Here we present selected details of the emulators built during the Reified Sequential Emulation for our example in section 5.8.

Here is the linear model fitted to the coarse runs used for building the emulator for $A(\theta^1, z^1, H^1)$. The data used for fitting this model is presented in figure E.5 The variables *t1*, *t2* and *zt* are the same as they were for the example of chapter 4. The variable C represents H^1, and the variable Vw represents our parametrization of the decision of whether or not to build and how much to run the model, labelled in the text via κ. We fit a saturated cubic model, having noticed that the corresponding quadratic fit left a significant pattern in the residual plots.

```
 Call:
lm(formula = Eloss ~ I(zt) + I(C) + I(Vw) + I(t2) + I(t1) + I(zt^2) +
    I(C * zt) + I(C^2) + I(Vw * zt) + I(Vw * C) + I(Vw^2) + I(t2 *
    zt) + I(t2 * C) + I(t2 * Vw) + I(t2^2) + I(t1 * zt) + I(t1 *
    C) + I(t1 * Vw) + I(t1 * t2) + I(t1^2) + I(zt^3) + I(C *
    zt^2) + I(C^2 * zt) + I(C^3) + I(Vw * zt^2) + I(Vw * C *
    zt) + I(Vw * C^2) + I(Vw^2 * zt) + I(Vw^2 * C) + I(Vw^3) +
    I(t2 * zt^2) + I(t2 * C * zt) + I(t2 * C^2) + I(t2 * Vw *
    zt) + I(t2 * Vw * C) + I(t2 * Vw^2) + I(t2^2 * zt) + I(t2^2 *
    C) + I(t2^2 * Vw) + I(t2^3) + I(t1 * zt^2) + I(t1 * C * zt) +
    I(t1 * C^2) + I(t1 * Vw * zt) + I(t1 * Vw * C) + I(t1 * Vw^2) +
    I(t1 * t2 * zt) + I(t1 * t2 * C) + I(t1 * t2 * Vw) + I(t1 *
    t2^2) + I(t1^2 * zt) + I(t1^2 * C) + I(t1^2 * Vw) + I(t1^2 *
    t2) + I(t1^3), data = fasta)


Residuals:
      Min        1Q    Median        3Q       Max
-1.173006 -0.054148 -0.001522  0.052593  1.497908


Coefficients:
```

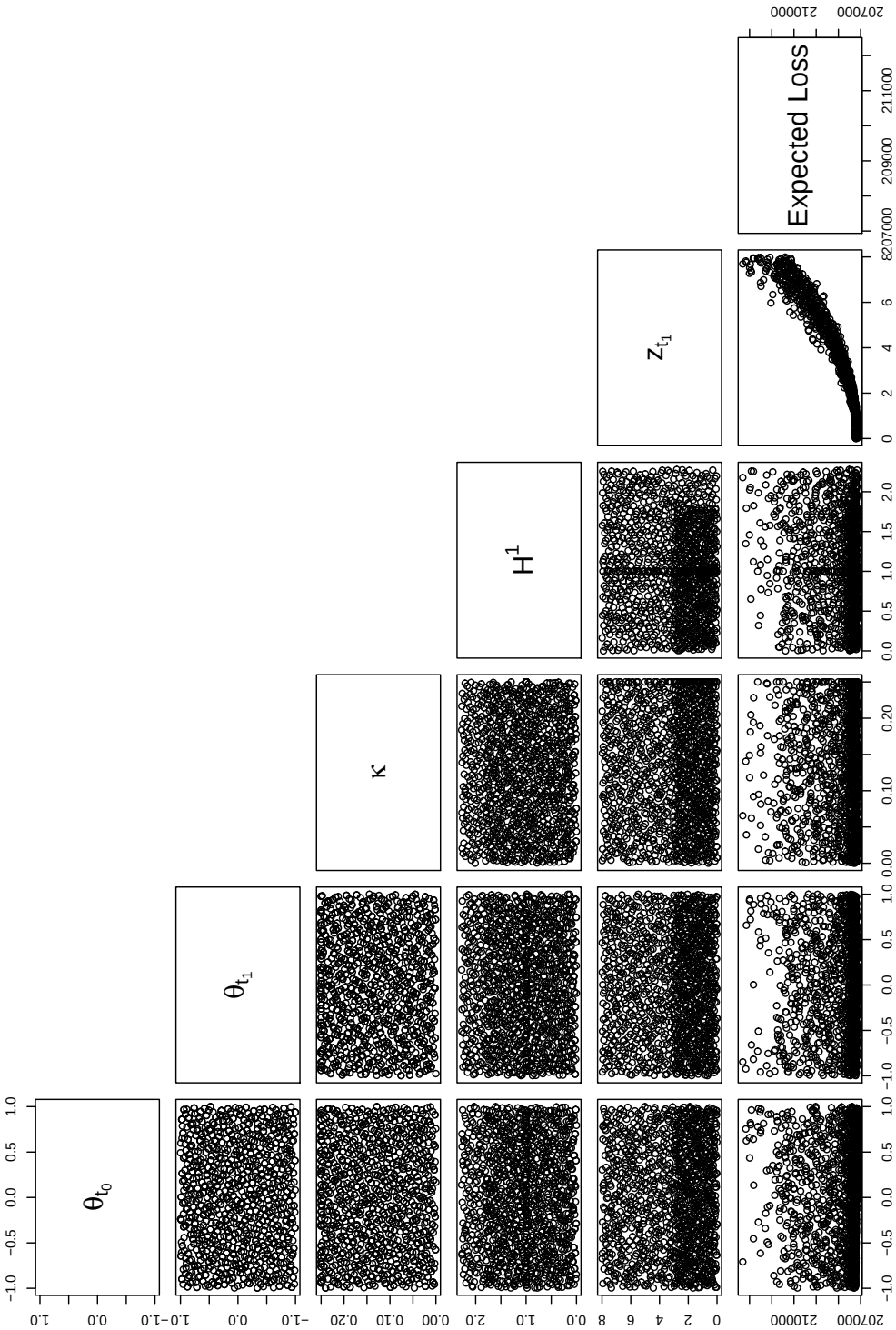

Figure E.5: The coarse data used for building an emulator for $A(\theta^1, z^1, H^1)$.

```
                 Estimate Std. Error   t value Pr(>|t|)
(Intercept)      3.296e-01  5.516e-02     5.975 3.07e-09 ***
I(zt)           -3.879e-01  2.528e-02   -15.342  < 2e-16 ***
I(C)            -1.351e-01  9.035e-02    -1.495 0.135113
I(Vw)            1.496e+00  8.134e-01     1.839 0.066139 .
I(t2)           -7.117e-02  5.204e-02    -1.368 0.171702
I(t1)           -1.206e-01  5.061e-02    -2.384 0.017306 *
I(zt^2)          1.283e-01  5.515e-03    23.266  < 2e-16 ***
I(C * zt)        6.912e-02  1.643e-02     4.206 2.80e-05 ***
I(C^2)          -6.696e-03  7.155e-02    -0.094 0.925465
I(Vw * zt)       1.571e+00  1.438e-01    10.930  < 2e-16 ***
I(Vw * C)        2.781e-01  5.272e-01     0.528 0.597906
I(Vw^2)         -2.043e+01  5.619e+00    -3.637 0.000289 ***
I(t2 * zt)       3.867e-02  1.448e-02     2.671 0.007679 **
I(t2 * C)        9.855e-02  5.119e-02     1.925 0.054486 .
I(t2 * Vw)       3.499e-01  4.540e-01     0.771 0.441063
I(t2^2)         -3.524e-01  4.107e-02    -8.579  < 2e-16 ***
I(t1 * zt)       3.984e-02  1.451e-02     2.746 0.006127 **
I(t1 * C)       -1.906e-02  5.141e-02    -0.371 0.710965
I(t1 * Vw)      -3.447e-01  4.528e-01    -0.761 0.446696
I(t1 * t2)      -5.444e-02  3.705e-02    -1.469 0.142012
I(t1^2)         -4.253e-02  4.168e-02    -1.020 0.307841
I(zt^3)         -1.349e-03  4.418e-04    -3.053 0.002317 **
I(C * zt^2)      1.351e-02  1.487e-03     9.082  < 2e-16 ***
I(C^2 * zt)      3.291e-02  5.018e-03     6.559 8.21e-11 ***
I(C^3)           2.752e-02  1.991e-02     1.382 0.167098
I(Vw * zt^2)    -1.317e-02  1.223e-02    -1.077 0.281764
I(Vw * C * zt)  -1.239e+00  4.242e-02   -29.200  < 2e-16 ***
I(Vw * C^2)     -9.821e-01  1.572e-01    -6.247 5.90e-10 ***
I(Vw^2 * zt)    -6.606e-01  3.778e-01    -1.749 0.080599 .
I(Vw^2 * C)      1.294e+01  1.512e+00     8.558  < 2e-16 ***
```

```
I(Vw^3)          1.942e+01  1.343e+01     1.446 0.148509
I(t2 * zt^2)    -1.239e-03  1.592e-03    -0.778 0.436459
I(t2 * C * zt)  -1.762e-06  5.322e-03 -0.000331 0.999736
I(t2 * C^2)     -2.875e-02  2.005e-02    -1.434 0.151913
I(t2 * Vw * zt)  2.889e-03  4.331e-02     0.067 0.946819
I(t2 * Vw * C)  -1.803e-01  1.676e-01    -1.076 0.282285
I(t2 * Vw^2)    -8.415e-01  1.490e+00    -0.565 0.572229
I(t2^2 * zt)     3.605e-01  6.382e-03    56.485  < 2e-16 ***
I(t2^2 * C)      5.719e-02  2.348e-02     2.436 0.015005 *
I(t2^2 * Vw)     2.890e-01  1.916e-01     1.508 0.131741
I(t2^3)         -4.083e-02  2.808e-02    -1.454 0.146252
I(t1 * zt^2)     1.484e-02  1.587e-03     9.354  < 2e-16 ***
I(t1 * C * zt)  -8.343e-03  5.191e-03    -1.607 0.108306
I(t1 * C^2)     -1.009e-02  2.005e-02    -0.503 0.614764
I(t1 * Vw * zt)  1.527e-01  4.357e-02     3.506 0.000473 ***
I(t1 * Vw * C)   4.266e-01  1.689e-01     2.526 0.011686 *
I(t1 * Vw^2)    -1.232e+00  1.478e+00    -0.834 0.404683
I(t1 * t2 * zt)  7.502e-02  5.632e-03    13.320  < 2e-16 ***
I(t1 * t2 * C)  -7.752e-03  2.091e-02    -0.371 0.710912
I(t1 * t2 * Vw) -4.915e-02  1.704e-01    -0.288 0.773137
I(t1 * t2^2)     2.444e-01  2.477e-02     9.868  < 2e-16 ***
I(t1^2 * zt)     6.234e-02  6.269e-03     9.943  < 2e-16 ***
I(t1^2 * C)     -2.131e-02  2.341e-02    -0.911 0.362745
I(t1^2 * Vw)     4.230e-02  1.881e-01     0.225 0.822125
I(t1^2 * t2)     7.886e-03  2.453e-02     0.322 0.747847
I(t1^3)         -3.434e-02  2.810e-02    -1.222 0.222001
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.146 on 1144 degrees of freedom
Multiple R-squared: 0.9958,Adjusted R-squared: 0.9956
```

```
F-statistic:  4951 on 55 and 1144 DF,  p-value: < 2.2e-16
```

The first thing we notice is that the main effects are in z_{t_1}, which is the same thing we noticed in the main example from chapter 4 and was expected having seen the data in figure E.5. The new variables also appear to have some effect on the analysis. For example, the interaction between z_{t_1}, H^1 and κ appears to have a significant impact on our expected loss, particularly if z_{t_1} is large and we have not run the new model very much. We checked that the large R^2 values were merely an artifact of having had to emulate our expected loss over an artificially large range of z_{t_1} in the same way as we did in section E.2.

We noted from the residual plots in figure E.6 that there may have been some heteroscedasticity in the residuals against z_{t_1}. We handled this in the same way as in previous Sequential Emulations and modelled the prior variance of the correlated residual on both the coarse and accurate models as an increasing function of z_{t_1}. We chose the same values for the variances of the uncorrelated errors on both coarse and accurate models, for all three emulators, as we did for the example in Chapter 4. For the current emulator this meant a coarse uncorrelated component of the residual with variance 0.005, and a variance of 0.0005 for the uncorrelated component of the accurate residual. We used the same heuristic for fixing correlation length as we used throughout the Sequential Emulation performed in Chapter 4, giving correlation parameters with values 1.6,1.6,102.4,1.231148, and 0.1 for θ_{t_0}, θ_{t_1}, κ, H^1, and z_{t_1} respectively.

Here is the saturated cubic fit to the coarse runs used for building the emulator for $B^1_{\lambda^1}(\theta_{t_0}, z^1, H^1)$. We fitted a cubic model after noticing that the residuals from a quadratic fit contained a definite signal.

```
Call:
lm(formula = Eloss ~ I(zt) + I(C) + I(Vw) + I(t1) + I(zt^2) +
    I(C * zt) + I(C^2) + I(Vw * zt) + I(Vw * C) + I(Vw^2) + I(t1 *
    zt) + I(t1 * C) + I(t1 * Vw) + I(t1^2) + I(zt^3) + I(C *
    zt^2) + I(C^2 * zt) + I(C^3) + I(Vw * zt^2) + I(Vw * C *
    zt) + I(Vw * C^2) + I(Vw^2 * zt) + I(Vw^2 * C) + I(Vw^3) +
    I(t1 * zt^2) + I(t1 * C * zt) + I(t1 * C^2) + I(t1 * Vw *
```

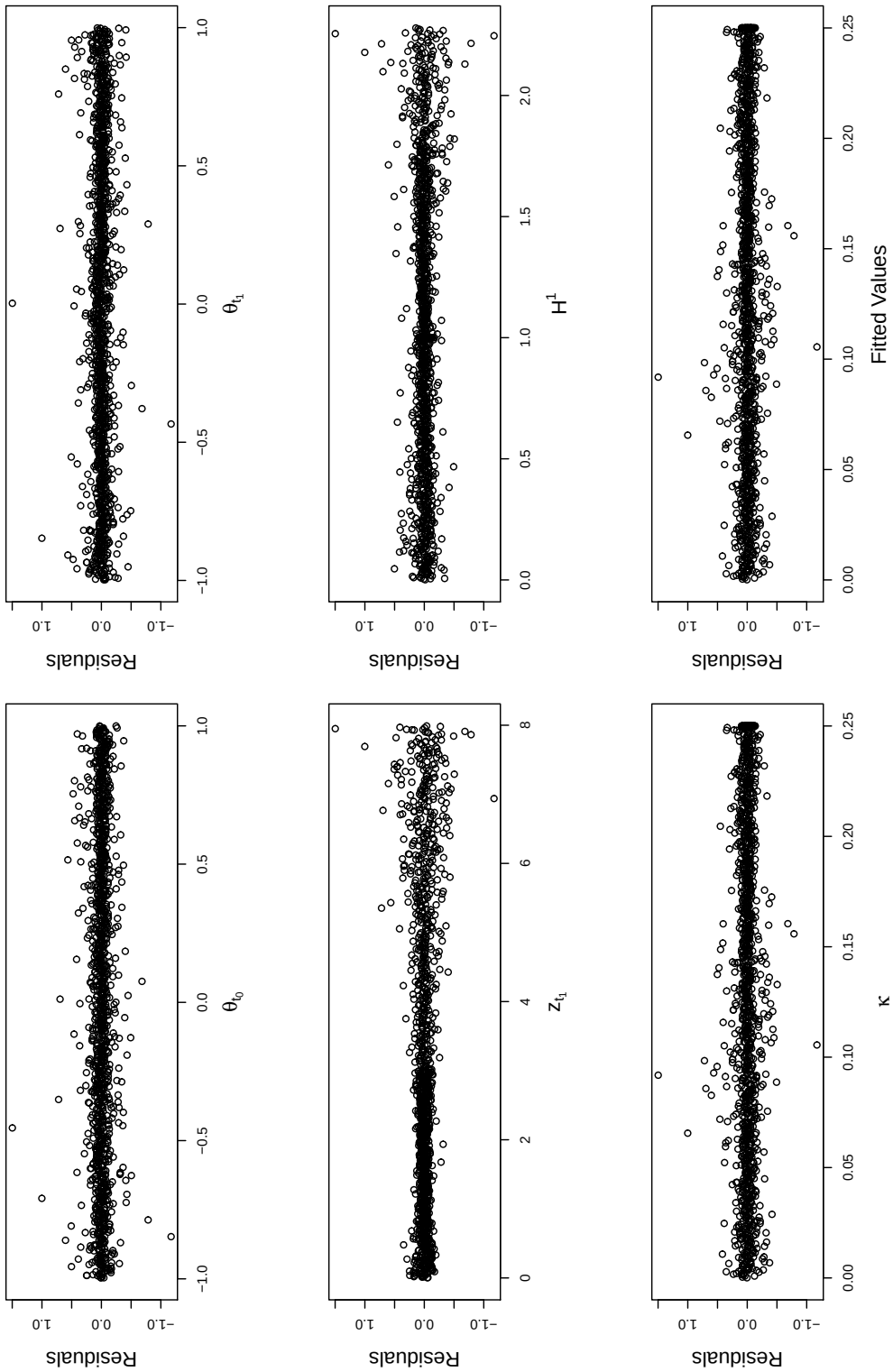

Figure E.6: Residual plots from fitting a saturated linear model of order three to the coarse data for $A(\theta^1, z^1, H^1)$.

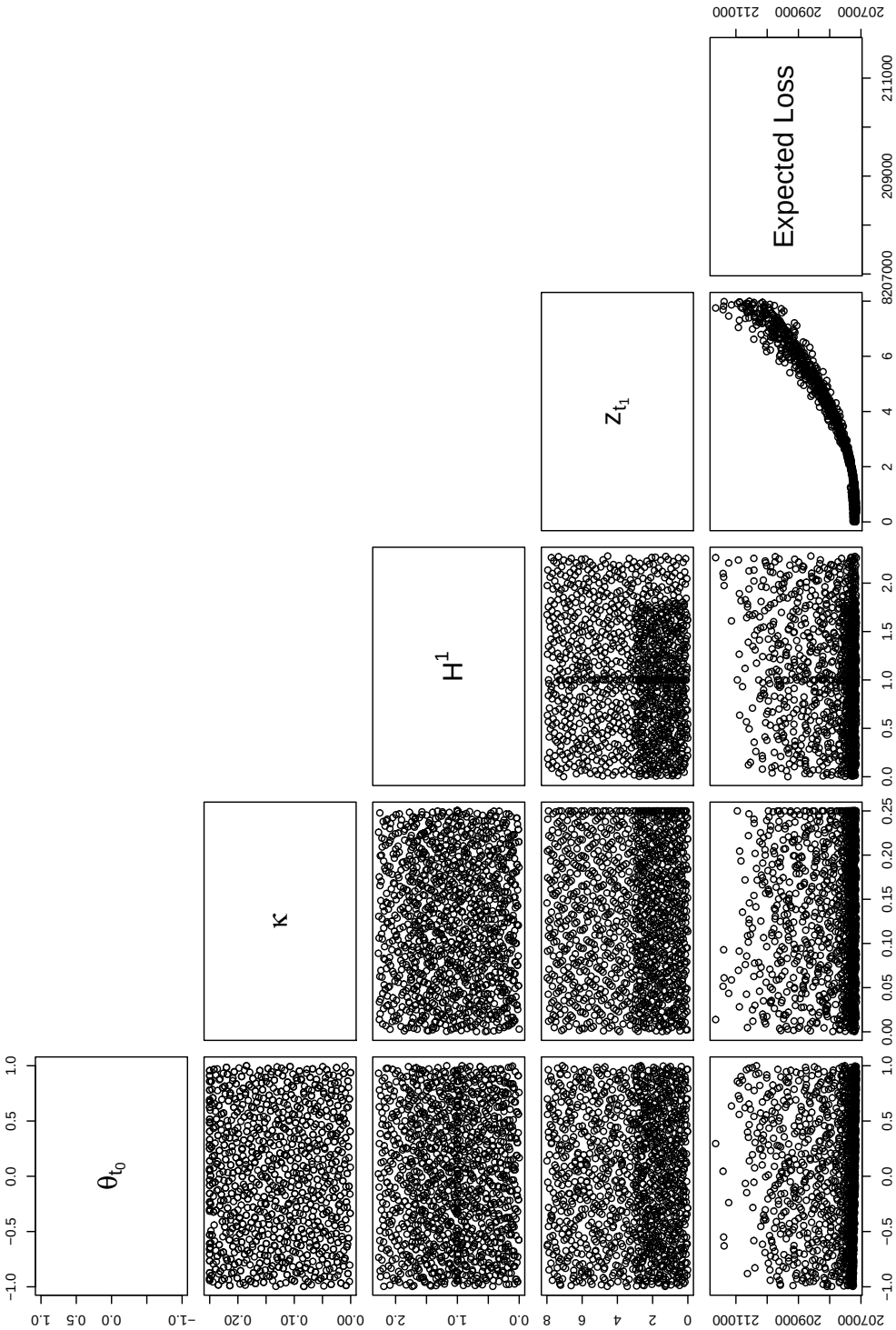

Figure E.7: The coarse data used for building an emulator for $B^1_{\lambda^1}(\theta_{t_0}, z^1, H^1)$.

```
    zt) + I(t1 * Vw * C) + I(t1 * Vw^2) + I(t1^2 * zt) + I(t1^2 *
    C) + I(t1^2 * Vw) + I(t1^3), data = fastb)

Residuals:
       Min         1Q     Median         3Q        Max
-1.0010905 -0.0485048 -0.0005665  0.0523429  1.5060290

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)      3.333e-01  5.388e-02   6.186 8.52e-10 ***
I(zt)           -3.674e-01  2.572e-02 -14.286  < 2e-16 ***
I(C)            -3.073e-01  9.078e-02  -3.386 0.000734 ***
I(Vw)            2.018e+00  8.127e-01   2.483 0.013150 *
I(t1)           -1.600e-01  5.073e-02  -3.155 0.001647 **
I(zt^2)          1.477e-01  5.621e-03  26.281  < 2e-16 ***
I(C * zt)        7.311e-02  1.683e-02   4.344 1.52e-05 ***
I(C^2)           1.382e-01  7.236e-02   1.909 0.056493 .
I(Vw * zt)       1.510e+00  1.466e-01  10.299  < 2e-16 ***
I(Vw * C)        8.006e-01  5.281e-01   1.516 0.129793
I(Vw^2)         -2.762e+01  5.673e+00  -4.869 1.28e-06 ***
I(t1 * zt)       5.667e-02  1.483e-02   3.820 0.000140 ***
I(t1 * C)        1.657e-03  5.153e-02   0.032 0.974357
I(t1 * Vw)      -3.688e-02  4.528e-01  -0.081 0.935098
I(t1^2)          3.724e-02  4.208e-02   0.885 0.376305
I(zt^3)         -1.757e-03  4.497e-04  -3.908 9.86e-05 ***
I(C * zt^2)      1.423e-02  1.526e-03   9.327  < 2e-16 ***
I(C^2 * zt)      4.504e-02  5.135e-03   8.772  < 2e-16 ***
I(C^3)          -1.891e-02  2.021e-02  -0.936 0.349680
I(Vw * zt^2)    -2.498e-02  1.247e-02  -2.003 0.045385 *
I(Vw * C * zt)  -1.438e+00  4.283e-02 -33.566  < 2e-16 ***
I(Vw * C^2)     -1.008e+00  1.599e-01  -6.305 4.08e-10 ***
```

```
I(Vw^2 * zt)     4.276e-01  3.842e-01   1.113 0.265979
I(Vw^2 * C)      1.280e+01  1.536e+00   8.328 2.29e-16 ***
I(Vw^3)          3.569e+01  1.356e+01   2.631 0.008622 **
I(t1 * zt^2)     1.465e-02  1.622e-03   9.037  < 2e-16 ***
I(t1 * C * zt)  -2.564e-02  5.374e-03  -4.772 2.05e-06 ***
I(t1 * C^2)     -1.245e-02  2.037e-02  -0.611 0.541240
I(t1 * Vw * zt)  1.335e-01  4.370e-02   3.054 0.002312 **
I(t1 * Vw * C)   5.498e-01  1.704e-01   3.227 0.001285 **
I(t1 * Vw^2)    -2.855e+00  1.491e+00  -1.915 0.055713 .
I(t1^2 * zt)     2.024e-02  6.352e-03   3.187 0.001475 **
I(t1^2 * C)     -3.983e-02  2.368e-02  -1.682 0.092903 .
I(t1^2 * Vw)     7.927e-02  1.895e-01   0.418 0.675829
I(t1^3)          6.776e-03  2.849e-02   0.238 0.812035
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.1487 on 1165 degrees of freedom
Multiple R-squared: 0.9957,Adjusted R-squared: 0.9956
F-statistic:  7914 on 34 and 1165 DF,  p-value: < 2.2e-16
```

Again, we notice that the main effects are in z_{t_1}, but there are non-negligible effects in the other variables. We also checked that the large value of R^2, which is an artifact of the strong signal in z_{t_1}, did not mask different behaviour in the other variables in the same way as we have earlier in this appendix. We used the same methods of fixing the parameters required for the Bayes Linear multi-level update in the same way as we did for the emulator of $A(\theta^1, z^1, H^1)$ described above.

The linear model used to help build our emulator for $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0}, H^1)$ is shown below.

```
Call:
lm(formula = Eloss ~ I(Vw) + I(t1) + I(Vw^2) + I(t1 * Vw) + I(t1^2) +
    I(Vw^3) + I(t1 * Vw^2) + I(t1^2 * Vw) + I(t1^3), data = fastc)
```

```
Residuals:
       Min         1Q     Median         3Q        Max
-0.0196273 -0.0049774 -0.0001106  0.0051463  0.0233583

Coefficients:
              Estimate Std. Error  t value Pr(>|t|)
(Intercept)   0.112895   0.000942  119.850  < 2e-16 ***
I(Vw)         0.125694   0.027974    4.493 7.70e-06 ***
I(t1)        -0.141022   0.001340 -105.202  < 2e-16 ***
I(Vw^2)      -1.052742   0.251115   -4.192 2.97e-05 ***
I(t1 * Vw)    0.369318   0.019056   19.381  < 2e-16 ***
I(t1^2)       0.115528   0.001371   84.248  < 2e-16 ***
I(Vw^3)       4.014156   0.640269    6.269 5.06e-10 ***
I(t1 * Vw^2) -1.385099   0.071178  -19.460  < 2e-16 ***
I(t1^2 * Vw) -0.209628   0.009106  -23.020  < 2e-16 ***
I(t1^3)       0.059153   0.001349   43.847  < 2e-16 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1   1

Residual standard error: 0.007074 on 1190 degrees of freedom
Multiple R-squared: 0.9858,Adjusted R-squared: 0.9857
F-statistic:  9171 on 9 and 1190 DF,  p-value: < 2.2e-16
```

By observing the data used to fit this model, shown in figure E.8, we would expect terms involving θ_{t_0} to be responsible for a large proportion of the variation and we can see this in our model and in the picture of the final emulator presented in figure 5.3. We used the same methods as we have for our previous multi-level emulations to fix the required parameters.

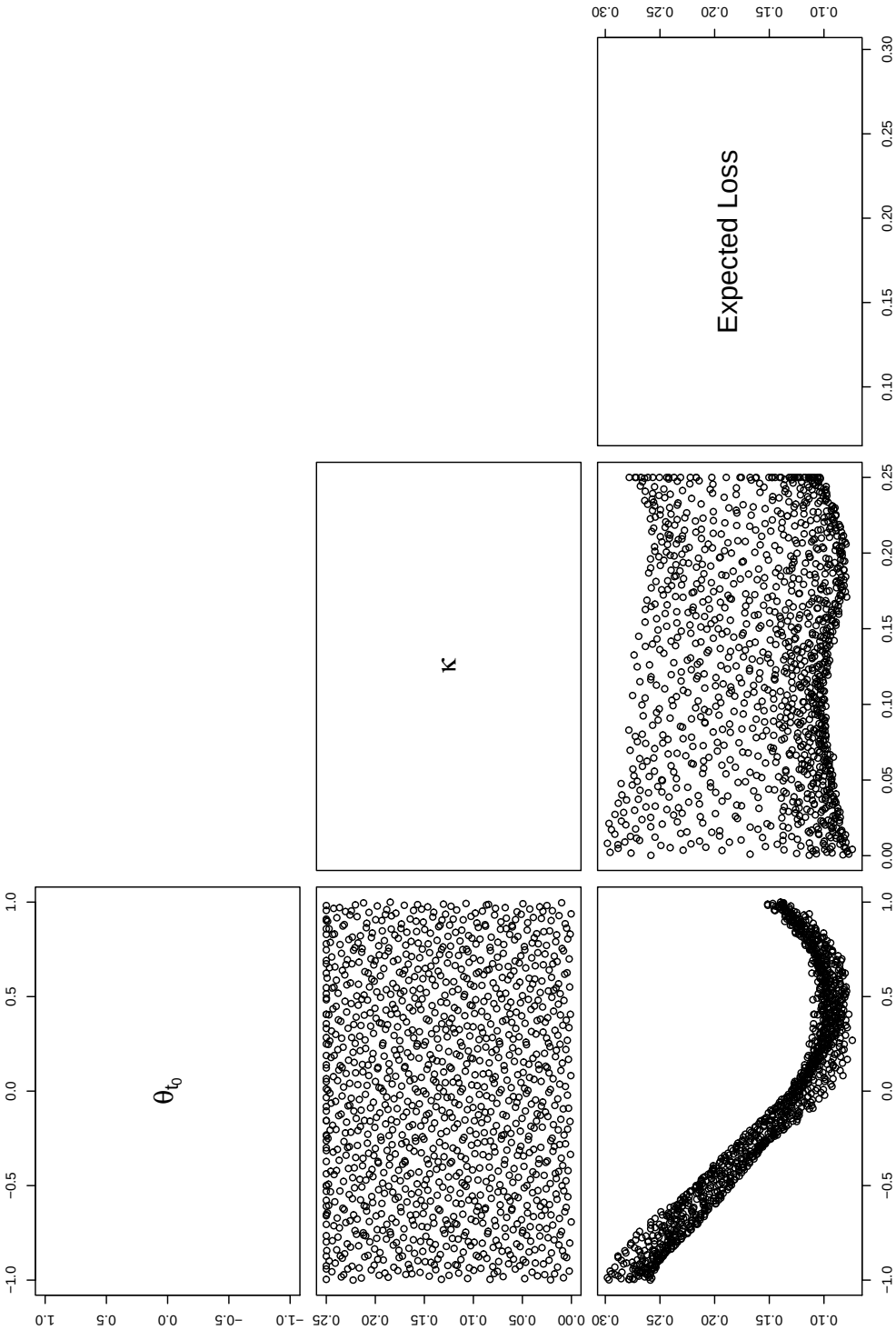

Figure E.8: The coarse data used for building an emulator for $C^1_{\lambda^1}(\theta_{t_0}, z_{t_0}, H^1)$.

Appendix F

Computing the hat function moments with a computer algebra package

Here we present some of the code developed for using the computer algebra package Maple [70] to express the hat function moments in terms of expectations depending only on functions of x^*, θ, $u_{t_0}(x^*)$, β, η, e_{t_0} and $u(\hat{x}, \theta)$, conditioned on x^*, for any form of $g(\cdot, \cdot)$.

There are a number of tasks that must be completed by the computer algebra package before we can emulate the hat function moments accurately. Our goal is to generate expressions for $E\left[\hat{f}(\theta)|x^*\right]$, $Var\left[\hat{f}(\theta)|x^*\right]$, $Cov\left[\hat{f}(\theta), y|x^*\right]$ and $Cov\left[\hat{f}(\theta), e_{t_0}|x^*\right]$, each of which will be the sum of expectations of the variables above. Some of these expectations may be evaluated immediately using our prior specification and will leave simple functions of θ. Others can be expressed as expectations of functions of x^*, conditioned on x^*, that we can then pass back to another program in order to evaluate them numerically by integrating out x^*. Expectations of functions of $u(\hat{x}, \theta)$ would also be passed into another program so that they could be emulated.

The first task handled by the computer algebra package is to define the rules that govern the expectation operator, and to define the covariance function in terms of expectations. To do this we define *function_variables* which are functions about

which we are uncertain and therefore have a non-constant expectation. We define the expectation operator so that the expectation of a *function_variable* b is returned as $E(b)$, and so that the operator is linear.

```
`type/function_variable`:=proc(t)
     member(t,function_variables) end:
`type/function_expr`:=proc(t)
     type(t, algebraic) and hastype(t, function_variable) end:
`type/scalar_expr` := proc(t)
     type(t,algebraic) and not hastype(t, function_variable) end:
E:='E': define(E,E(a::nonunit(algebraic) + b::nonunit(algebraic))=E(a)+E(b),
              E(a::nonunit(scalar_expr)*b::function_expr)=a*E(b),
              E(a::nonunit(scalar_expr))=a,
              E(b::function_expr*a::nonunit(scalar_expr))=a*E(b),
              E(a::nonunit(scalar_expr)*E(b::nonunit(function_expr)))=a*E(E(b)));
```

The covariance of two expressions is defined via $Cov\left[a,b\right]=E\left[ab\right]-E\left[a\right]E\left[b\right]$.

```
cov:= proc(left,right,function_variables)
local expr, first, second;
expr:= left*right;
expr:=expand(expr);
first:=E(expr);
second:=E(left)*E(right);
first - second;
end:
```

We now establish the quantities that appear in the calculation. v and W are the vector v and matrix W that are part of the definition of $\hat{x}$; u is the vector of model emulator residuals; B is the matrix of emulator coefficients, β; *eta* is the vector of discrepancies, η; e is the system observation error, e; *mu* is $\hat{x}$; and *fhat* is $\hat{f}(\theta)$. The function *trivArrays* takes as inputs the dimensions of x and θ; the length of $g(x,\theta)$; the number of simulator outputs, h, corresponding to historical values of the system for which we have observations; and the number of simulator outputs, f, corresponding to future values of the system. The function defines each of the quantities mentioned above as vectors and matrices of the correct dimension, and establishes the fact that u will be a vector of functions each having two inputs.

```
function_variables:=[]:
v:=NULL:
W:=NULL:
eta:=NULL:
e:=NULL:
B:=NULL:
u:=NULL:
mu:=NULL:
fhat:=NULL:

trivArrays := proc(v,W,B,eta,e,u,h,f,glength,xlength,declength)
 local i;
 v:=array(1..xlength,1..1);
 W:=array(1..xlength,1..h);
 B:=array(1..(f+h),1..glength);
 eta:=array(1..h,1..1);
 e:=array(1..h,1..1);
 u:=array(1..(f+h),1..1);
 for i from 1 by 1 to (f+h) do
  define(u[i,1]);
 od:
end:
```

In order to perform the hat function calculations we need a function that establishes the form of $g(x, \theta)$. Our code accepts any function that returns a vector of the correct length, and is usually given as the input *gfun*. An example of the format that is required for the function *gfun* is given by the function below.

```
g := proc(x,dec,n,m,row:=true)
local L1,L2,L3,i,j,transg;
with(linalg);
L1:=[1];
L2:=[NULL];
for i from 1 by 1 to n do
L2:= [op(L2),x[i,1]];
od:
L3:=[NULL];
for j from 1 by 1 to m do
```

```
L3:=[op(L3),dec[j,1]];
od:
transg:=array(1..1,1..(1+n+m),[[op(L1),op(L2),op(L3)]]);
if row
then RETURN(array(1..1,1..(1+n+m),[[op(L1),op(L2),op(L3)]]));
else RETURN(transpose(transg));
fi;
end:
```

We now compute $\hat{x}$. This involves constructing the correct expression for z_{t_0} out of the other variables and then performing the vector calculation $v + Wz_{t_0}$.

```
xhat := proc(dec,gfun,glength,x,xlength,declength,h,v,W,eta,e,B,u,mu,
                        function_variables)
 local Bh, Bhg, k, l,uh,uhx,z;
 with(linalg):
 Bh := array(1..h,1..glength);
 for k from 1 by 1 to h do
  for l from 1 by 1 to glength do
   Bh[k,l] := B[k,l];
  od:
 od:
 uh := proc(t)
  array(1..h,1..1,[seq([u[j,1](t)],j=1..h)]);
 end:
 uhx:=uh(x);
 function_variables:=seq(uhx[j,1],j=1..h);
 Bhg:=evalm(Bh&*gfun(x,dec,xlength,declength,false));
 z:=evalm(Bhg+uhx+eta+e);
 mu:=evalm(v+evalm(W&*z)):
end:
```

Having computed the expression for $\hat{x}$, we compute $\hat{f}(\theta)$ directly by plugging $\hat{x}$ (mu) in the place of x in our expression for the model emulator. A subtle point to note about this function and the last is that we are building the vector of *function_variables* as we go. This means that the order of these computations is crucial.

```
Fhat := proc(dec,gfun,glength,x,xlength,declength,h,f,v,W,eta,e,B,u,mu,
                        function_variables)
```

```
 local uf, ufxhat,trend;
 with(linalg);
 uf := proc(t,s)
  array(1..(f+h),1..1,[seq([u[j,1](t)],j=1..h),seq([u[k,1](t,s)],k=(h+1)..(f+h))]);
 end:
 ufxhat := uf(mu,dec);
 function_variables:=[op(function_variables),seq(ufxhat[j,1],j=1..(f+h))];
 function_variables:=op(function_variables);
 trend:=evalm(B&*gfun(mu,dec,xlength,declength,false));
 RETURN(evalm(trend+ufxhat));
end:
```

The function *Fhat* expands the expression

$$\beta g(v + W(\beta_{t_0 j} g_j(x^*, 0) + u_{t_0}(x^*) + \eta_{t_0} + e_{t_0}), \theta) + u(\hat{x}, \theta)$$

into a linear combination of functions of each of the variables and correctly defines which variables, when conditioned on x^*, remain uncertain. We now take expectations in order to obtain $E\left[\hat{f}(\theta)|x^*\right]$.

```
ExpectFhat := proc(h,f,gfun,glength,xlength,declength,x,dec,function_variables,
                                    v,W,B,eta,e,u,mu,fhat)
local i,tBs,fvs;
 tBs:=[NULL];
 for i from 1 by 1 to glength do
   tBs := [op(tBs),seq(B[j,i],j=1..(f+h))];
 od;
 fhat:=Fhat(dec,gfun,glength,x,xlength,declength,h,f,'v','W','eta','e','B','u',
                      'mu','function_variables');
 for i from 1 by 1 to (f+h) do
   fhat[i,1]:=expand(fhat[i,1]);
 od;
 fvs:=op(function_variables);
 function_variables:=[op(fvs),op(tBs),seq(eta[j,1],j=1..h),seq(e[j,1],j=1..h)];
 RETURN(array(1..(f+h),1..1,[seq([E(fhat[j,1])],j=1..(f+h))]));
end:
```

We can then use the covariance function to obtain $Var\left[\hat{f}(\theta)|x^*\right]$

```
 #Only to be done after ExpectFhat so that function_variables is correct
VarFhat:= proc(h,f,gfun,glength,xlength,declength,x,dec,function_variables,v,W,
                              B,eta,e,u,mu,fhat)
   array(1..(f+h),1..(f+h),[seq([seq(cov(fhat[j,1],fhat[k,1],
                            function_variables),k=1..(f+h))],j=1..(f+h))]);
end:
```

Here we construct the vector y as a linear combination of functions of x^*, θ, $u(x^*, \theta)$, β, η. We do this in order to calculate $Cov\left[\hat{f}(\theta), y\right]$ conditioned on x^*.

```
thisY:=proc(h,f,gfun,glength,xlength,declength,x,dec,function_variables,v,W,B,
                       eta,e,u)
  local uf,ufx,trend;
  uf := proc(t,s)
    array(1..(f+h),1..1,[seq([u[j,1](t)],j=1..h),
               seq([u[k,1](t,s)],k=(h+1)..(f+h))]);
  end:
  ufx := uf(x,dec);
  function_variables := [op(function_variables),seq(ufx[j,1],j=1..(f+h))];
  function_variables := op(function_variables);
  trend := evalm(B&*gfun(x,dec,xlength,declength,false));
  RETURN(evalm(trend+ufx+eta));
 end:
```

We may now express $Cov\left[\hat{f}(\theta), y|x^*\right]$ as a linear combination of expectations of functions of our uncertain variables.

```
CovFhatY:=proc(h,f,gfun,glength,xlength,declength,x,dec,function_variables,v,W,
                              B,eta,e,u,mu,fhat)
  local tY;
  tY := thisY(h,f,gfun,glength,xlength,declength,x,dec,'function_variables','v',
  'W','eta','e','u');
  for i from 1 by 1 to (f+h) do
    tY[i,1]:=expand(tY[i,1]);
  od;
  RETURN(array(1..(f+h),1..(f+h),[seq([seq(cov(fhat[j,1],tY[k,1],
                              function_variables),k=1..(f+h))],j=1..(f+h))]));
end:
```

We do the same for $Cov\left[\hat{f}(\theta), e_{t_0}|x^*\right]$ here.

```
CovFhate:=proc(h,f,e,fhat,function_variables,x,dec,v,W,B,eta,u,mu)
 array(1..(f+h),1..h,[seq([seq(cov(fhat[j,1],e[k,1],function_variables),
                                                k=1..h)],j=1..(f+h))]);
end:
```

The following is an example of how we might execute the code for a particular size of model and length of $g(x, \theta)$.

```
h:=2;
f:=1;
x:=array(1..2,1..1);
theta:=array(1..1,1..1);
glength:=4;
xlength:=2;
declength:=1;
trivArrays('v','W','B','eta','e','u',h,f,glength,xlength,declength):
xhat(theta,g,glength,x,xlength,declength,h,'v','W','eta','e','B','u','mu',
   'function_variables'):
EFhat:=ExpectFhat(h,f,g,glength,xlength,declength,x,theta,'function_variables',
                    'v','W','B','eta','e','u','mu','fhat'):
VFhat:=VarFhat(h,f,g,glength,xlength,declength,x,theta,'function_variables','v',
                    'W','B','eta','e','u','mu','fhat'):
CFhatY:=CovFhatY(h,f,gfun,glength,xlength,declength,x,theta,'function_variables',
                    'v','W','B','eta','e','u','mu','fhat'):
CFhate:=CovFhate(h,f,'e','fhat','function_variables',x,theta,'v','W','B','eta',
                    'u','mu'):
```

The rest of our code assumes that $g(x, \theta)$ contains only products of monomials in x and θ. Here we use the package *combinat* to set all expectations of products containing at least one element of each of the two distinct sets $\{\beta, u_{t_0}(x^*), u(\hat{x}, \theta)\}$ and $\{\eta, e_{t_0}\}$ to be zero, and expectations of functions of both η and e_{t_0} to be zero as well. We need to specify the variable *gOrder*, which is the highest power of x in any monomial in $g(x, \theta)$.

```
with(combinat):
ForCombs:=[]:
for i from 1 by 1 to ((4*gOrder)-2) do
 temp:=op(ForCombs):
```

```
 ForCombs:=[temp,op(function_variables)]:
od:
for i from 1 by 1 to ((2*h)+f+(glength*(f+h))) do
 for j from (1+(2*h)+f+(glength*(f+h))) by 1 to ((4*h)+f+(glength*(f+h))) do
  E(function_variables[i]*function_variables[j]):=0:
  startSeq:=[function_variables[i],function_variables[j]]:
  for k from 1 by 1 to ((4*gOrder)-2) do
   combs:=choose(ForCombs,k):
   for l from 1 by 1 to nops(combs) do
    newSeq:=[op(startSeq),op(combs[l])]:
    E(mul(newSeq[i],i=1..(2+k))):=0:
   od:
  od:
 od:
od:
```

We now assume that β has a multivariate normal distribution and determine all of its higher order moments as functions of it's mean and variance, by using the characteristic function for the Normal distribution. We first define the characteristic function then establish the mean and variance of β as the arrays *tMu* and *tSig*. The rest of the code changes the expectations of all functions of the elements of β alone into functions of *tMu* and *tSig*. We could then input the correct numbers for these matrices here or in another program to simplify our expressions for the hat function moments.

```
CharFun := proc(t,tMu,tSig)
 local first, second;
 with(linalg);
 first := evalm(transpose(t)&*tMu):
 second := evalm(transpose(t)&*tSig&*t):
 exp((I*(expand(evalm(transpose(t)&*tMu)))[1,1]) -
      (0.5*(expand(evalm(transpose(t)&*tSig&*t)))[1,1])):
end:

t:=array(1..(glength*(f+h)),1..1):
tMu:=array(1..(glength*(f+h)),1..1):
tSig:=array(1..(glength*(f+h)),1..(glength*(f+h))):
```

```
BetaSet:=[seq(function_variables[j],j=(h+h+f+1)..(h+h+f+(glength*(h+f))))]:
ForMoments:=[];
for i from 1 by 1 to (4*gOrder) do
 temp := op(ForMoments):
 ForMoments := [temp, op(BetaSet)]:
od:
BCombs := [];
for i from 1 by 1 to (4*gOrder) do
 temp := op(BCombs):
 temps := choose(ForMoments,i):
 BCombs := [temp,op(temps)]:
od:
print(nops(BCombs));
CFun := CharFun(t,tMu,tSig):
DiffSet:=array(1..nops(BCombs));
for i from 1 by 1 to nops(BCombs) do
 tSet:=[]:
 for j from 1 by 1 to nops(BCombs[i]) do
  for k from 1 by 1 to (h+f) do
   for l from 1 by 1 to glength do
    temp:=op(tSet):
    if
      is(B[k,l] in BCombs[i][j])
      then
      tSet := [temp,t[k+((h+f)*(l-1)),1]]:
    fi;
   od;
  od;
 od;
 DiffSet[i] := diff(CFun,op(tSet)):
od:
for i from 1 by 1 to (glength*(h+f)) do
  t[i,1]:=0:
od:
for i from 1 by 1 to nops(BCombs) do
E(mul(BCombs[i][j],j=1..nops(BCombs[i]))) := (1/(I^(nops(BCombs[i]))))*DiffSet[i]:
od:
```

At this point we would then write the output out to a file and use R, say, to read the output in and perform the numerical integrations with respect to x^* and to emulate those functions that depend on $u(\hat{x}, \theta)$.