Optimising Arrival Management in Air Traffic Control





Lingyi Yang
Pembroke College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy
Hilary 2022

Acknowledgements

I would like to start by thanking my supervisors, Sam and Jari, for their endless patience and support over the last 4 years. Our meetings were packed with ideas and exotic teas, as well as interesting trivia and other advice — memories that I will treasure. It was an honour to be a part of the indomitable horde.

Rich, Marc, John, Ben, Liam, Jonathan, and James at NATS welcomed me as a part of the team and provided fascinating background to the industrial problem arising from air traffic control.

A shout out to all my collaborators in the Datasıg team, as well as those who navigated through the DECOVID project with me.

My life as a DPhil student was made more colourful by the plentiful spontaneous discussions over coffee and cake in the common room. I thank my fellow cohort 4s, and the wider InFoMM family for the camaraderie and cheer. In particular, Ambrose and Victor provided endless office entertainment — mathematics or otherwise. Olly and Giancarlo extended the breadth of innumerable conversations. Zhen helped me stay grounded when reality and ideals seemed irreconcilable. Helen shared fantastic photos and memes that made me smile on tough days. Ellen was a voice of reason and calm.

Finally, I would like to thank my family, who encouraged and supported me throughout my life.

Thank you all for your company in this rewarding journey during its most arduous times.

Abstract

Efficient landing is a key component of improving air transport, both for passengers' experience and fuel consumption. With just two runways, Heathrow airport was running at 98% capacity before the pandemic. The existing queuing system allows buffer time for the aircraft to land on one runway, but this can add delays to journeys and be fuel inefficient. With the recovery of the travel industry after the 2020 pandemic, improving landing procedures remains a pertinent problem to NATS (who manages all the air traffic in UK airspace). In this thesis, we develop alternative methods to sequence aircraft as they approach for landing at Heathrow.

In the first part of this thesis, we cast the arrival management problem in a reinforcement learning framework. We design a basic air traffic model and apply both table representation methods and nonlinear approximation with a neural network. Specifically, we compare the performance of Q-learning, SARSA, and DDPG on this environment. Further we explore dimension reduction/feature representation through path signatures. Finally we design multi-grid inspired neural network structures and see that these lead to faster training but ultimately comparable performance.

For the second part, we look at the problem from a different perspective. We take inspiration from the theory of optimal transport and formulate an entropy-regularised optimisation problem. We design an algorithm with block gradient descent-like steps and note that the conflicts in the set-up of our problem introduces non-convexity even when working in the (convex) space of distributions over arrival times. By adding an additional 'considerate' cost, akin to a Pigouvian tax, the performance of the algorithm is enhanced.

Finally, the last part of this thesis shows the flexibility of our approach. We adapt our work to apply to a new air traffic design concept being researched at NATS.

Contents

1 Industrial motivation and background **8**
- 1.1 Introduction . . . 8
- 1.2 Industrial Motivation . . . 8
  - 1.2.1 Arrival management in Air Traffic Control . . . 9
  - 1.2.2 Arrival Managers (AMANs) . . . 10
  - 1.2.3 New airspace design and concept – streaming at metering points . . . 11
- 1.3 Mathematical motivation . . . 12
- 1.4 Structure of the thesis . . . 14

I Reinforcement Learning **15**

2 Background **16**
- 2.1 Reinforcement Learning . . . 16
  - 2.1.1 Markov decision processes (MDPs) . . . 17
  - 2.1.2 Model-free methods – prediction problem . . . 20
  - 2.1.3 Model-free methods – control problem . . . 23
  - 2.1.4 Approximate methods . . . 25
  - 2.1.5 Policy gradients . . . 28
  - 2.1.6 Actor-critic methods . . . 30
- 2.2 Path Signatures . . . 32
  - 2.2.1 Definition and properties of the signature . . . 33
  - 2.2.2 Signatures of data and their transforms . . . 36
- 2.3 Multigrid methods . . . 38
  - 2.3.1 Concept of multigrid methods . . . 39
  - 2.3.2 Multiscale methods in image analysis . . . 42
  - 2.3.3 Recent multigrid methods in machine learning . . . 43

3 Arrival management as a deep reinforcement learning problem **44**
- 3.1 Our approach . . . 44
- 3.2 The model . . . 45
- 3.3 Results . . . 47
  - 3.3.1 Comparison with table representations . . . 47
  - 3.3.2 Restricted frequency of controls . . . 48
  - 3.3.3 Restricted number of controls . . . 50
  - 3.3.4 Results with path signature features . . . 53
  - 3.3.5 Multigrid-inspired RL . . . 54

3.4 Stability of deep RL methods . . . 56
3.4.1 Summary of the key methods and theoretical results in the RL community . . . 56
3.4.2 Regularisation . . . 61

4 Summary **62**

II Optimising (air) transport with optimal transport theory **63**

5 Background **64**
5.1 Motivation . . . 64
5.2 Optimal Transport . . . 64
5.2.1 Introduction . . . 64
5.2.2 The regularised problem . . . 67
5.3 Continuous Optimisation . . . 69
5.3.1 Introduction . . . 69
5.3.2 Gradient projection methods . . . 70
5.3.3 Coordinate descent methods . . . 71

6 An optimal transport inspired approach to Air Traffic Control **73**
6.1 Air Traffic Control Motivation . . . 73
6.1.1 Monge Formulation . . . 73
6.1.2 Kantorovich Formulation . . . 74
6.2 A row-wise gradient algorithm . . . 77
6.2.1 Global gradient descent . . . 77
6.2.2 Row-wise gradient steps . . . 78

7 Economic Interpretation **83**
7.1 Background . . . 84
7.1.1 Welfare Economics . . . 84
7.1.2 Prisoner's dilemma . . . 85
7.1.3 Social good and externalities . . . 86
7.2 Pareto optimality in air traffic control . . . 89
7.2.1 Two plane case . . . 89
7.2.2 General case . . . 89

8 Multi-agent games **91**
8.1 A simple 2 player, 2 action game . . . 91
8.1.1 Considerate cost formulation . . . 94
8.2 Generalising the 2 player case to M actions . . . 97
8.3 Generalising to N agents . . . 102
8.4 Numerical results for multi-agent anti-coordination games . . . 103

9 Considerate costs for arrival management problem **104**
9.1 Considerate cost formulation . . . 104
9.2 Numerical experiments . . . 107
9.3 Real data example . . . 109

10 Summary **113**

III A decentralised approach **114**

11 Arrival streaming over metering points **115**
11.1 Arrival streaming . . . 115
11.1.1 Modified AMAN – Under prototyping . . . 116
11.2 Modification to the transport problem set-up . . . 118
11.3 AMOPT algorithm applied to simulation data . . . 119
11.3.1 Scale of the arrival streaming problem . . . 120
11.3.2 The simulation data . . . 121
11.3.3 Updating the minimum and maximum ETO times . . . 121
11.3.4 The results on simulation data . . . 122
11.4 Discussion . . . 125
11.5 Summary . . . 126

12 Conclusion and extensions **127**
12.1 Summary of the thesis . . . 127
12.2 Extensions . . . 128
12.2.1 Reinforcement learning for arrival streaming . . . 129
12.2.2 Multi-agent reinforcement learning (MARL) . . . 129
12.3 Future airspace research . . . 130

Bibliography **132**

Appendices **146**

A Dynamic programming **146**

B Optimal Control Theory **149**
B.1 The classical PDE approach . . . 149
B.1.1 Finite horizon . . . 150
B.1.2 Infinite horizon . . . 151
B.1.3 Bellman's Principle of Optimality . . . 151
B.1.4 The Hamilton–Jacobi–Bellman (HJB) equation . . . 152
B.2 Viscosity solutions . . . 153

C Derivation of the HJB equation (infinite horizon case) **156**

D Hyperparameters **158**
D.1 Toy model . . . 158
D.2 Other models using 100 intervals . . . 159

E Multigrid inspired neural networks **160**

F Linear quadratic regulator example **162**
F.1 Monotonicity . . . 162
F.2 1D continuous LQ problem . . . 163
F.3 1D discrete LQ problem . . . 169
F.3.1 Q-Learning . . . 170
F.3.2 DPG . . . 173

G Proof of Proposition 8.1.3 **174**

H Proof of Proposition 8.2.1 **183**

I Numerical results for multi-agent anti-coordination games **187**
I.1 Two-actions . . . 187
I.2 Many actions . . . 191
I.3 Many agents . . . 192

J Further work on signatures and Neural CDEs **195**
J.1 Online neural CDEs . . . 195
J.2 Applications of signatures in data science for healthcare . . . 195
J.2.1 Early detection of sepsis . . . 196
J.2.2 Early warning score differences in patients with and without covid . . . 197

Chapter 1

Industrial motivation and background

1.1 Introduction

Every time we make a journey in a commercial aircraft, we rely on air traffic controllers to ensure the safety and punctuality of our flight. This DPhil project is in collaboration with NATS, the company that manages all of the air traffic control for UK upper airspace, as well as at many airports, including Heathrow [4]. There are exciting new proposals and planned policies for the management of aircraft, which would significantly disrupt the status quo. As a result, new methods need to be researched and designed.

One constraint on passenger capacity at an airport is the availability of the runway(s). A particularly busy region in the UK airspace is the section above London. One reason for this is that Heathrow, one of the busiest airports in the world, has just two runways — usually only one of which is used for landing. To ensure that the runway for landing is fully utilised, there are four holding stacks which act as queues in which arriving aircraft can circle to wait. Successive aircraft need to be sufficiently separated for safety reasons as they approach for landing, and there are different separation rules based on the weights of pairs of consecutive aircraft. Therefore, sequencing aircraft based on size can further increase runway utilisation. These queues are vital in giving air traffic controllers time to sequence landings.

The current tool to aid air traffic controllers in reordering aircraft performs a simplistic search over a few discrete options. This thesis explores whether we can give instructions to planes further in advance so that they arrive in a sequenced manner, and can land almost immediately. This way, waiting times can be eliminated or significantly reduced, thereby increasing the fuel efficiency of flights which has both financial and environmental implications.

In this chapter, we give a thorough overview of the industrial problem, and some of the mathematical considerations. In addition, we outline the rest of this thesis.

1.2 Industrial Motivation

In this section we set out the arrival management problem at Heathrow. The Europe-wide program, *Single European Sky ATM Research (SESAR)* [7], aims to unify and overhaul existing practices so that efficiency within the air traffic management industry is increased. In view of these impending changes, NATS are looking to implement various process improvements.

One such change will be dividing the control of air traffic into two regimes: *free route* and *systemised*. This means that above a certain height, in the free route regime, there will be minimal management of flights as long as they pass specific locations. This contrasts with the current practice, where there are defined routes for flights to follow. In lower airspace, under systemised control, there are still strict paths to follow and planes cannot overtake one another. Thus, in order to maximise efficiency of arrivals to airports, it is key to apply effective controls on aircraft in the transition from free route airspace to systemised airspace as they are approaching to land.

Since NATS manages all air traffic in the controlled airspace around the UK, it is important to plan and predict how this new system should work. In particular, it is vital to design the system so that capacity is increased (as has been the general trend prior the pandemic in 2020).

1.2.1 Arrival management in Air Traffic Control

The busiest parts of the UK airspace are the regions around London. This area contains several major airports, including Heathrow, which is one of the busiest airports in the world with almost half a million take-offs and landings per year (pre-pandemic). In Table 1.1 we see data from 2017 on the 9th to 13th busiest airports according to number of movements (take-offs and landings). Heathrow is the 12th busiest in terms of total number of take-offs and landings. However, when we look at the number of runways each of these airports has, it is apparent that resources at Heathrow are far more limited. In fact, before the pandemic, Heathrow was operating at 98% of its capacity [3]. Heathrow has just two runways, making it the busiest two-runway airport in the world. Aside from a short period in the morning, where both runways can be utilised for arrivals to accommodate the transatlantic arrival rush, most of the time, only one of these runways can be used for landing. As a result, arrival management is particularly critical — the constraint on runway availability means that arrivals need to be carefully planned so they slot in at exactly the right time. This also means that air traffic controllers operate under a high level of stress in order to maintain such performance.

Rank	Airport	# Movements	# Passengers	# Runways
9	Amsterdam - Schiphol	514,625	68,515,425	6
10	Shanghai - Pudong	496,774	70,001,237	5
11	Paris - Charles de Gaulle	482,676	69,471,442	4
12	London - Heathrow	475,915	78,012,825	2
13	Frankfurt	475,537	64,500,386	4

Table 1.1: Statistics from 2017 on the 9th to 13th busiest airports in the world according to the total number of take-offs and landings. Each take-off or landing counts as one movement. Taken from Airports Council International [1].

In Figure 1.1 we see the actual radar tracks of flights above London on a typical morning. Some clusters are formed, these are *holding stacks*, which are queues that allow planes to circle around until they are cleared for landing. Incoming aircraft must sometimes be stacked due to traffic congestion, poor weather, or runway unavailability. An airport may have several holding stacks; depending on flight routes and runways. Heathrow has four holding stacks, called Bovingdon, Lambourne, Ockham, and Biggin.

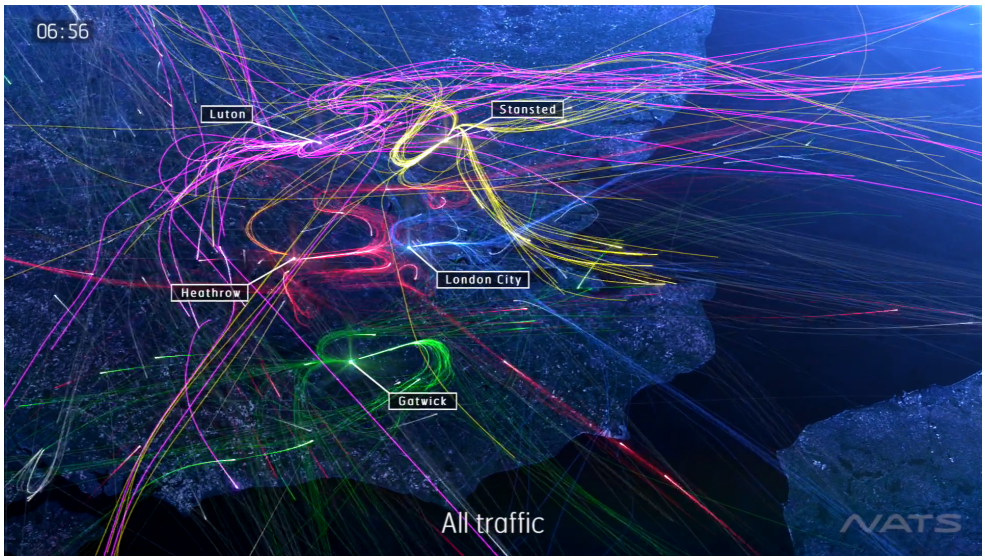


Figure 1.1: Radar tracks of aircraft arriving at London airports on a typical pre-pandemic morning. Credit: NATS.

The planes waiting in the stacks are not necessarily cleared to land on a first-come-first-served basis. This is because minimum separation distances between consecutive aircraft are based on their weight category (which is a discrete classification). All aircraft cause turbulence in the air, known as wake vortices. However, larger, heavier aircraft cause more disturbances. Lighter aircraft are more easily affected by turbulence so we must leave a greater separation gap between a heavy and a light aircraft as they approach for landing. This is essential as wake vortices are extremely hazardous and have led or contributed to fatal crashes [8].

To increase utilisation of the runway, we want to reorder the arriving aircraft so that similar-sized planes arrive together and time between successive arrivals are minimised. Currently, the four holding stacks create buffer time for controllers to arrange arrivals [87]. However, this system also contributes to delays in journeys and wastes fuel, which has both financial and environmental implications.

The key question is whether instructions can be given to aircraft further in advance, so that they arrive in a sequenced manner and can land almost immediately without the need for queues.

1.2.2 Arrival Managers (AMANs)

Arrival Managers (AMANs) are tools to help aid sequencing of aircraft; there are many different versions in use in air traffic control. Their main functionalities are similar.

For sequencing, AMANs usually perform a simple search over a small number of options. As the space of arrival times is continuous, and the resulting optimisation problem is difficult, each aircraft is associated with a number which is its landing order. The sequence of landing order is discrete and easier to optimise, hence AMAN amends this instead. Separately, the minimum separation distance between different aircraft types is computed based on additional data such as the weather, in particular if visibility is affected. Finally AMAN constructs a target landing time for each aircraft from the landing sequence and required minimum separation distance so that consecutive aircraft maintain minimum separation if

they achieve their target times. It is then the air traffic controllers' job to issue instructions so that these target times are met as closely as possible. Note that although AMAN contains trajectory prediction, the target times are only functions of the landing order and the weight categories and are not directly optimised by the AMAN. That is, in conventional AMANs, the trajectory prediction function is not used in the sequencer [177].

The objective function that is used to change the landing sequence is given by (see also [58], [148], and [26])

$$L_{\text{total}} = L_{\text{delay}} + L_{\text{exchange}} + L_{\text{trail}}.$$

The delay penalty, L_{delay}, is the sum of the differences between the target arrival times and the scheduled time of arrival from the flight plan of each aircraft. This is ultimately the key objective to minimise.

The exchange penalty L_{exchange} penalises changes from the *first-come-first-served (FCFS)* order. The purpose of this component is to ensure that any changes to the sequence should provide sufficient benefits in the reduction of delays. If a flight is not at the FCFS order in the landing sequence, then there is a penalty for each swap required to transform the sequence to the FCFS sequence, which is precisely the Kendall tau distance [103].

Finally the in-trail penalty L_{trail} penalises swapping flights which are on the same route. This is standard in modelling and optimisation of traffic flow management. To be in 'trail order', flights must be on the same route and their separation is less than 30 nautical miles (this criterion can vary depending on airspace and local configuration). Each time a sequence changes from the original 'in-trail order', there is a penalty, which is set as the exchange penalty multiplied by some multiple/factor (the magnitude of this factor can be adjusted based on how many swaps violating the in-trail order is worth to air traffic controllers).

The question for us is: can we provide a more realistic, flexible system that can directly optimise the target time of arrival instead of working with a landing sequence? By extending the region of control, what instructions can we give to approaching planes, so that they arrive in a sequenced manner, and can land almost immediately with significantly reduced hold times [45]? Understanding and improving algorithms for this purpose forms the main goal of this thesis.

1.2.3 New airspace design and concept – streaming at metering points

One key performance indicator for NATS is how many aircraft can achieve what is known as *continuous descent*. This is where an aircraft can descend at a suitable rate without restrictions (which are required if there are other aircraft flying at a lower height).

Achieving continuous descent has multiple benefits, for one, fuel efficiency is increased if they can descend at a constant rate. Saving fuel is very important to limit the contribution to climate change, as well as hitting environmental targets at NATS. In addition, fuel is a high-cost consumable, and airlines already operate on very tight profit margins, so any savings would be greatly appreciated. Additionally, there may also be a lower workload on the air traffic controllers if they need to communicate less with the aircraft.

Motivated by this, a new idea that is in the research and testing phase at NATS is the concept of *arrival streaming at metering points*. Metering points are a set of existing waypoints along current routes. They can be repurposed so that additional controls can be imposed relative to each metering point to ensure minimum separation is maintained at these points.

For all aircraft at all times, minimum separation distance has to be maintained for safety. For en-route separation, this is typically either a 5 nautical mile horizontal separation or a

vertical separation of 1000 feet (this may be adjusted based on weather and visibility). This distance can be achieved through either vertical or horizontal separation. Since the current practice does not target sequencing at the metering points, vertical separation is commonly used. As a result, aircraft can arrive at metering points simultaneously (but on different flight levels).

As safety is paramount in air traffic control, the lack of safe longitudinal separation leads to the requirement to ensure a safe vertical separation. Having aircraft around similar longitudinal positions across different flight levels means that continuous descents are not possible. Aircraft have to descend in a stepwise fashion, decreasing fuel efficiency, and also increasing air traffic controller workload.

A sufficient condition for continuous descent is that the aircraft are longitudinally separated at the metering points. Having 90 seconds of difference between arrival times (to metering points) for successive aircraft is a generous rule-of-thumb to ensure longitudinal separation. By ensuring adequate time-separation at the metering points, continuous descent can be achieved.

1.3 Mathematical motivation

In this section we highlight the mathematical considerations of the industrial problem discussed above. As is the case with many applied/industrial problems, we first need to determine some simplifying assumptions to create a mathematical model.

The industrial problem asks: how can we manoeuvre the aircraft in the system such that there is minimum overall delay. Broadly speaking, this is an optimisation problem. We have the objective of minimising delays given physical constraints in the form of the speed of aircraft, how the instructions can be issued and so on.

For smaller systems, if we only consider the absolute order of the flights, it is possible to perform an exhaustive search. We need to impose constraints so that the new ordering of aircraft is feasible. This is essentially what the existing AMAN system attempts, though a reshuffled order for the flights does not directly correspond to actions the air traffic controllers should make. For large and complex systems, more efficient methods are required.

To solve an optimisation problem, we need to define an objective and constraints. There are several approaches to modelling this problem. Although the industrial motivation is derived from continuous time and continuous space, we can model this as a discrete or continuous system. There is a host of techniques in the existing literature for both choices. For continuous models, we can view this as a stochastic control problem [196] (with uncertainties arising for example in the measurement of data). Using the dynamic programming principle, we can then derive the Hamilton–Jacobi–Bellman (HJB) equation — a second order PDE. These are typically solved numerically. For discrete systems, there are methods from classical dynamic programming [20] or reinforcement learning [173].

For the action/control space, if we choose to send out one instruction per aircraft, then the number of controls will vary with the number of aircraft currently in the system. This naïvely corresponds to changing the dimensionality of the action space, and results in a difficult problem. In a discrete setting (where each aircraft has a discrete location), we can overcome this issue by sending out an instruction per discrete interval. This, however, means the dimensionality of the control space scales with the resolution of our discretisation.

We can design a discrete toy model of the sequencing problem and apply reinforcement learning methods. We model the air traffic controller as an agent that observes the number

of aircraft and where they are, then sends out speed instructions. If an aircraft lands safely without violating minimum separation constraints then our agent gains a positive reward. Through repeated simulations, the agent can learn the actions which leads to an increase in the sum of expected rewards.

As we take finer discretisations of the problem, the dimensionality of the action space increases. Very quickly, cataloguing all possible scenarios becomes computationally prohibitive. In order to resolve this issue, we find a reduced representation of our model, a function approximation. Neural networks are nonlinear function approximators that have performed extremely successfully for high-dimensional reinforcement learning problems; we use them to represent the states of our problem.

Classical reinforcement learning methods have convergence guarantees (under certain conditions) which ensure they yield globally optimal strategies. By using neural networks as function approximators, we lose these guarantees and are likely to find a locally optimal strategy instead. We show in this thesis that this is indeed the case — the function approximation gives a fast initial increase in the performance when training policies, but this eventually converges to a local optimum (Q-learning and SARSA converge to a higher policy value on the same small toy problem).

Once we have determined the model, another challenge is deciding which features of the state space are useful to represent our problem. As mentioned above, it is more efficient to have long sequences of aircraft of similar sizes. This means that if we plot aircraft size against distance from the airport then the preferred configuration should have long flat regions. Therefore, we appeal to mathematical techniques that incorporate geometric properties such as these. We can consider the plot of aircraft size against distance as a path. Path signatures [119] capture the roughness of paths. Therefore intuitively these signatures capture key geometric information that we can use as features if we need to reduce the dimension of the problem.

Path signatures are invariant to reparameterisation which is useful as we may discretise with a different number of intervals. Since our problem is inherently spatial, we may also look at multi-scale methods. In general solving the problem with finer discretisation is computationally harder, there has been research looking at multi-grid method inspired neural network— this is also something we explore in this thesis.

In the arrival management problem, there is an initial distribution of aircraft arrivals over time (we can think of the distribution of each aircraft as deterministic at their estimated time). We need to map this initial distribution of arrival times to another distribution, which would have a better objective value. This is closely related to the theory of optimal transport. Under the Monge formulation, particles are mapped to another location, no particle splitting is allowed. This requirement is relaxed in the Kantorovich formulation. As a result, the problem is reduced to a linear programming problem. By adding an entropy regulariser, a target measure can be computed efficiently using the Sinkhorn algorithm. Inspired by optimal transport theory, we can formulate the arrival management problem as an entropy regularised optimisation problem.

Instead of modelling the air traffic controller as an agent, we can model each aircraft as an agent. Each agent is trying to optimise their objectives with the range of actions available to them — an n-player game. When the objective function is separable, block coordinate descent can significantly reduce the dimension of the optimisation problem. Convergence results are limited for non-convex problems, and often there is no guarantee we are converging even to a local optimum. Inspired by the theory of externalities in economics, we construct a 'considerate cost' for our optimisation problem; this can alleviate problems observed with

selfish agents and is analogous to a Pigouvian tax.

1.4 Structure of the thesis

This thesis is divided into three parts. In Part I, we cast the arrival management problem in a reinforcement learning framework. The theory required for this part is outlined in Chapter 2. In particular, this chapter includes an overview of reinforcement learning concepts and terminologies, a brief survey of reinforcement learning methods, signatures as features/representation of a path object, and an introduction to multigrid methods.

In Chapter 3 we present one way of framing the arrival management problem at Heathrow as a reinforcement learning problem by designing a discrete toy model that simulates basic aircraft arrival.

Using this custom environment, we obtain results of running both more traditional reinforcement learning methods like Q-learning, as well as newer deep reinforcement learning methods like DDPG. We also perform experiments of dimension reduction with signature features as well as altering the structure of the neural network inspired by multigrid methods. However, we observe some instabilities in these experiments, and go on to discuss and survey stability and convergence results in reinforcement learning, and the use of regularisation. Chapter 4 concludes and summarises this part.

In Part II we formulate the arrival management problem instead as an entropic regularised optimisation problem, with ideas from optimal transport.

Similar to the first part, we start with the necessary background materials in Chapter 5. This then leads us into Chapter 6 where we give an alternative formulation (compared to Part I) of the arrival management problem, and solve this by optimising a 'transport matrix' with coordinate-descent like steps and the use of an entropy regulariser.

An issue that can be seen with our first algorithm, is that to avoid conflicts (aircraft that are too close together), we cannot simply increase the magnitude of the parameter for the conflict cost in the objective function. We see that this is because this term is non-convex. Some insight and motivation from economic theory to address this issue, in particular Pareto optimality and externalities, are discussed in Chapter 7. In Chapter 8 we perform analysis on two-player, multi-action games, and derive conditions on a 'considerate cost' which would ensure convergence to the social optimum. For additional insight, we also formulate an algorithm for the general n-player case.

Motivated by this additional analysis of considerate cost, we formulate an analogous considerate cost for the air traffic control problem in Chapter 9. We give the results on real data from NATS, and show that introduction of such a cost can bring additional benefits. Part II concludes with Chapter 10, which contains a summary of the work in this part.

The final part, Part III, examines a slightly different problem — distributed arrival management over metering points. The problem formulations in Parts I and II are centralised, the key objective is sufficient separation at Heathrow. A modified decentralised problem involves optimising the flow of traffic at several points, known as metering points, which are further away from Heathrow. In Chapter 11, we discuss the necessary modifications and apply the tools we developed in Part II on some simulation data from NATS.

Finally in Chapter 12, we conclude and summarise the contributions of this thesis. By bringing together the concepts discussed and giving an overview to the multi-agent reinforcement learning literature, we discuss how this work on arrival management in air traffic control may be extended.

Part I

Reinforcement Learning

Chapter 2

Background

In this chapter we outline some prerequisites to our modelling approaches for the arrival management problem in air traffic control.

We can think of the air traffic controller as an agent that needs to take actions to achieve the objective of maintaining separation as aircraft approach Heathrow. This can be cast as a *control/reinforcement learning* problem, with the actions as speed instructions, and the reward function based on achieving the goal of correctly achieving separation.

For high-dimensional problems of interest, the reward landscape can often be quite flat, in the sense that it is difficult to distinguish between the performance of different controls, or highly non-convex [116]. It may be useful to find a subspace/extract features so that we actually work with a problem of lower dimension. When sequencing aircraft at Heathrow and considering wake vortices, a key property of the aircraft is their size. Dimension reduction techniques should still encode the ordering of aircraft and their sizes. The sequence of aircraft arrivals can be viewed as a *path*. *Path signatures*, motivated by rough path analysis, have become popular as features since they can capture relative ordering in a path.

When solving a discretised spatial problem, solving the problem on multiple scales and feeding the solution of the coarser methods to a finer discretisation can lead to accelerated convergence [195]. As we address our spatial problem with discrete methods, we can utilise these concepts in our methods.

For the rest of this chapter, we describe reinforcement learning, path signatures, and multigrid methods. Knowledge of these topics will be relied upon in this part of the thesis. For readers familiar with these concepts, this chapter is useful for establishing a standard set of notations that we use.

2.1 Reinforcement Learning

In 2015 we saw machines that can be trained to play Atari games better than humans [123]. Another significant development has been AlphaGo [164], a program designed by Deepmind, which defeated a top-level player at the game Go in 2016. Go is a far more complicated game than chess, in terms of the number of possible states, and hence the victory of AlphaGo over Lee Sodol was unprecedented. Both of these applications rely on deep reinforcement learning methods. Reinforcement learning is often considered to be the third branch of machine learning, together with supervised and unsupervised learning. Supervised learning is applied typically to classification or prediction problems, where there is a supervisor dictating the correct label/class. Unsupervised learning does not require natural labels; its

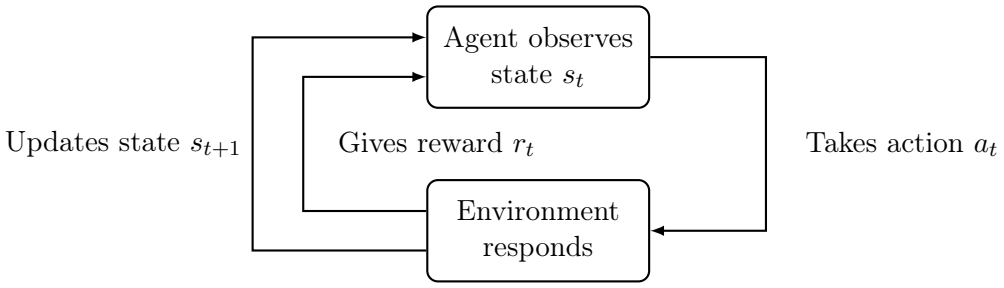


Figure 2.1: The interaction between an agent and its environment in one timestep.

aim is to find patterns in data. Reinforcement learning methods use a reward function to encourage an agent to learn desirable behaviour.

In this section, we review some background material on reinforcement learning. A key reference is [173]. We discuss the mathematical motivation and development from dynamic programming. A variety of methods are examined, some are classical while others are more recent developments based on deep neural networks.

For the rest of this section, we adopt the convention of using uppercase letters to denote random variables and lowercase letters to denote particular realisations of the random variables. A reinforcement learning problem is a discrete problem which consists of a state space $\mathcal{S}$ and an action space $\mathcal{A}$. For the majority of problems, there may only be a restricted set of actions that can be applied given a particular state s, we denote the set of feasible actions in this case as $\mathcal{A}(s)$. Where we do not explicitly specify a domain for the action, it is assumed to be the set of all feasible actions. The system evolves in discrete timesteps t, and an action can be chosen at each time.

The behaviour of the agent, that is, the way an action is chosen, is known as the policy. In the deterministic case, a policy maps a state to its corresponding action

$$\pi : \mathcal{S} \to \mathcal{A}, \text{ that is, } \pi(s) = a.$$

In the stochastic case, the policy describes the probability of choosing actions given the state

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1], \text{ that is, } \pi(a|s) = P(A_t = a|S_t = s).$$

As mentioned above, the key difference between reinforcement learning and the other two branches of machine learning is that the desired behaviour of what the agent should do is passed through a reward function. The typical scheme of reinforcement learning can be seen in Figure 2.1. At each timestep the agent observes the current state s_t and takes action a_t, the environment responds to this action and the agent receives a reward r_t (note that some sources e.g. [173] use the notation r_{t+1} instead for this reward, both are common in the literature) and observes the new state s_{t+1}. A model of how the environment responds to actions may be a component of the reinforcement learning problem, though it is not necessary in all approaches.

2.1.1 Markov decision processes (MDPs)

Markov decision processes (MDPs) are central in discrete-time control theory and have been used for over 60 years [91]. They are often used for modelling problems in reinforcement learning. In an MDP, the states $(S_t)_{t\geq 0}$ form a Markov process; S_t captures all relevant prior

information. Although most problems in reinforcement learning are modelled as MDPs, the agent may not be able to observe these states directly, in which case we say this is a partially observed Markov decision process (POMDP).

For an MDP with a finite number of states and actions, the state transitions are given by the conditional probability

$$p(s', r|s, a) = P(S_{t+1} = s', R_t = r|S_t = s, A_t = a); \tag{2.1}$$

this encapsulates the environment dynamics. Knowing, or having an estimate for, (2.1) is what is meant by having a model – it is a description of how the environment responds to each state-action pair. We may derive other state transition probabilities, such as

$$p(s'|s, a) = P(S_{t+1} = s'|S_t = s, A_t = a) = \sum_r p(s', r|s, a), \tag{2.2}$$

$$r(s, a) = E[R_t|S_t = s, A_t = a] = \sum_r r \sum_{s'} p(s', r|s, a), \tag{2.3}$$

$$r(s, a, s') = E[R_t|S_t = s, A_t = a, S_{t+1} = s'] = \sum_r r \frac{p(s', r|s, a)}{p(s'|s, a)}. \tag{2.4}$$

These equivalent forms may be more useful in proving properties, see examples in [173].

The immediate reward that we receive after the action at time t, denoted R_t, can be a useful indicator, but we may want to consider the rewards thereafter as well. The sum of rewards is typically referred to as the *return*.

Definition 2.1.1. *The discounted return is defined as*

$$G_t = \sum_{i=t}^{\infty} \gamma^{i-t} R_i = R_t + \gamma G_{t+1}, \tag{2.5}$$

where γ is known as the discount factor.

Typically we have $\gamma \in [0, 1]$. If $\gamma = 0$, then this means our agent is very shortsighted and only the immediate reward matters. If $\gamma = 1$ then this implies that future rewards are as important as current ones. This may also be useful when we are less certain about future rewards as the discount factor reduces their importance. Just as in classical control theory, reinforcement learning deals with two types of problems: problems with a finite horizon (episodic) or problems with an infinite horizon (continuing). With episodic problems, there may be a different reward behaviour upon reaching the terminal state, known as the terminal reward.

To compare different policies, we introduce state and action values.

Definition 2.1.2. *The state value $V_\pi(s)$ is the expected return following policy π starting at state s,*

$$V_\pi(s) = E_\pi[G_t|S_t = s]. \tag{2.6}$$

Similarly the action value $Q_\pi(s, a)$, is the expected return starting at state s, taking action a, then following the policy π thereafter

$$Q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a]. \tag{2.7}$$

We may also refer to the state values as *V-values* and the action values as *Q-values*.

We wish to find a policy such that the expected return is maximised, this gives us the optimal value functions

$$V^*(s) = \max_\pi V_\pi(s), \quad Q^*(s,a) = \max_\pi Q_\pi(s,a).$$

An optimal policy π^* that generates these optimal values may not necessarily be unique. We can infer an optimal policy from the optimal action values by taking a greedy policy

$$\pi^*(a|s) = \begin{cases} 1 \text{ if } a = \operatorname{argmax} Q^*(s,a) \\ 0 \text{ otherwise,} \end{cases} \tag{2.8}$$

dealing with ties arbitrarily where necessary.

Before we proceed, let us first state some popular terminologies used in the field of reinforcement learning. If the environment dynamics (2.1) are known and we want to improve our policy then this setting is called the *planning problem*. The case where the environment dynamics are unknown and we have to interact to 'learn' information is referred to as the *learning problem*. There are typically two parts to a reinforcement learning problem, *policy evaluation* (also known as *prediction*), and *control* (learning the optimal policy through *policy improvement*). Depending on the way the policy is chosen, reinforcement learning methods are said to be *value-based*, *policy-based* or an *actor-critic*. A brief outline of each is given in the following sections but further details can be found in [173].

Recall that we can define the return in a recursive fashion

$$G_t = R_{t+1} + \gamma G_{t+1}.$$

Using this and the tower property of conditional expectations on (2.6), (2.7), we derive the Dynamic Programming Principle.

Theorem 2.1.3 (Dynamic Programming Principle (DPP) – Discrete case)**.** *For all $s \in \mathcal{S}$ and all $t \in \{1, \dots, T\}$, the value functions satisfy*

$$\begin{aligned} V_\pi(s) &= E_\pi[R_t + \gamma V_\pi(S_{t+1}) | S_t = s] \\ &= \sum_a \left(\pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V_\pi(s')] \right), \end{aligned} \tag{2.9}$$

and

$$\begin{aligned} Q_\pi(s,a) &= E_\pi[R_t + \gamma Q_\pi(S_{t+1}, \pi(S_{t+1})) | S_t = s, A_t = a] \\ &= E_\pi[R_t + \gamma V_\pi(S_{t+1}) | S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a)[r + \gamma V_\pi(s')]. \end{aligned}$$

Therefore the value functions are the sum of the expected immediate reward and the expected return from the new state. For more details, see [173]. Using the DPP, problems can be broken down into subproblems, so that the solution of the full problem should also be optimal for the subproblem. Methods using the DPP are typically said to be *model-based* (transitions of the MDP are modelled explicitly), see Appendix A for a brief overview of discrete dynamic programming, including two very important classes of methods — *value iteration* and *policy iteration*.

From the DPP, we have $V_\pi(s) = \sum_a \pi(a|s)Q(s,a)$. Since the optimal state value function is the maximum expected return starting at state s, this must equal the maximum return of taking any action from the state s, i.e.

$$V^*(s) = \max_a Q^*(s,a).$$

Likewise, the optimal action value can be broken down into the immediate reward and future rewards by assuming the optimal policy is followed from the new state:

$$Q^*(s,a) = E[R_t + \gamma V^*(S_{t+1})|S_t = s, A_t = a].$$

We can rearrange the above [173] to define recursive versions of the optimal state and action values.

Definition 2.1.4. *The Bellman optimality equations are given by*

$$\begin{aligned} V^*(s) &= \max_a E[(R_t + \gamma V^*(S_{t+1})|S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V^*(s')], \end{aligned} \tag{2.10}$$

$$\begin{aligned} Q^*(s,a) &= E[(R_t + \gamma \max_{a'} Q^*(S_{t+1}, a')|S_t = s, A_t = a] \\ &= \sum_{s',r} p(s', r|s, a) \left[r + \max_{a'} \gamma Q^*(s', a') \right]. \end{aligned} \tag{2.11}$$

Many reinforcement learning algorithms have update steps based on these equations. They are the discrete forms of well-known results in classical control theory, see Appendix B for more details on the perspective from control theory.

2.1.2 Model-free methods – prediction problem

The dynamics of the environment may not be known to us. The family of methods that do not utilise the state transition probabilities are known as *model-free* methods. Note that there is a distinction between model-free methods and using simulations to estimate the transition probability (then applying the methods from the previous section). There are model-free methods that tackle the prediction and control problems. Well-known classes of model-free methods are Monte Carlo (MC) methods and temporal difference (TD) methods.

We will see many algorithms with update steps of this form:

$$\text{New Estimate} \leftarrow \text{Old Estimate} + \text{Step Size}(\text{Target} - \text{Old Estimate}). \tag{2.12}$$

In this section we look at the methods associated with the prediction problem, since solving the prediction problem often forms a critical part of the control problem. Monte Carlo methods can be applied on problems of finite horizon, they learn from complete episodes of experience. An *episode* is one complete run of a finite horizon/episodic problem. The principle of MC methods is simple, the estimated state (or action) values are the mean returns seen from starting in that state (and taking that action), there is no bootstrapping (using other estimated values to improve our current estimate).

Suppose we observe samples of episodes while following π. There are two popular approaches to estimating value functions: first-visit Monte Carlo and every-visit Monte Carlo [21]. Let us look at the estimation of the state values (the action values follow analogously).

For first-visit Monte Carlo, suppose the first time that state s is visited in an episode is t_s. When we see this state for the first time, we increment a counter $N(s) \leftarrow N(s) + 1$. If s is visited, at the end of the episode we increment a 'total return' variable by the (discounted) return between t_s and the terminal time, $\Sigma(s) \leftarrow \Sigma(s) + G_{t_s}$. After reaching a fixed total number of episodes/simulations, our estimate of the state value function is the mean of returns: $V(s) = \Sigma(s)/N(s)$. Here $V(s)$ is an unbiased estimator for V_π and $V(s) \to V_\pi(s)$ as $N(s) \to \infty$. Every-visit Monte Carlo is similar except the counter is incremented every time state s is visited in an episode, and the sum of returns is incremented by the respective return from each visit. Note that the estimate in this case can be biased, since returns from visits in the same episodes are not independent. This bias will tend to zero as $N(s) \to \infty$. See [21] for more a detailed comparison between the two.

The usual sample mean formula can be indexed by the number of samples and written as

$$\mu_k = \frac{1}{k}\sum_{i=1}^{k} x_i = \frac{1}{k}(x_k + (k-1)\mu_{k-1}) = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1}). \tag{2.13}$$

We can use (2.13) to formulate incremental versions of the Monte Carlo algorithms, which update the estimate for $V_\pi(s)$ after each episode. For the first-visit case, the specific form of (2.12) is

$$V(s) \leftarrow V(s) + \frac{1}{N(s)}(G_{t_{N(s)}} - V(s)),$$

where $t_{N(s)}$ is the $N(s)$th time of visiting s. There is a slight modification for the every-visit case. Suppose state s has been visited $N(s) - n$ times so far and s is visited a total of n times in the next episode, then

$$V(s) \leftarrow V(s) + \frac{1}{N(s)}\big(G_{t_{N(s)}} + ... + G_{t_{N(s)-n+1}} - nV(s)\big).$$

We can think of this intuitively as making a small correction towards the direction of new information gained. Note that both of these Monte Carlo methods are of the form (2.12). For simplicity/brevity, we only refer to the first-visit Monte Carlo from now on, which has a step size of $1/N(s)$ and G_t as the target.

Note that we can also use a *learning rate* α, instead of $1/N(s)$, when we move to the control problem. In finding any optimal policy, the policies which generated the samples will change, resulting in non-stationary data. This is where the incremental forms will be very useful. We can also design the learning rate such that older samples will be weighted less.

The disadvantage of Monte Carlo methods is that each episode must terminate before an update can be made, by which time we will have visited many other states, resulting in high variance. An alternative is to sample a few steps from the current state and then use estimates for the remaining states. This is precisely bootstrapping, as described in Appendix A. Temporal difference (TD) methods make use of this idea.

Like dynamic programming, temporal difference methods use bootstrapping for incomplete episodes. The simplest temporal difference method is TD(0) [21] with the update rule

$$V(S_t) \leftarrow V(S_t) + \alpha(R_t + \gamma V(S_{t+1}) - V(S_t)), \tag{2.14}$$

where α is the step size, $R_t + \gamma V(S_{t+1})$ is known as the TD-target and

$$\delta_t = R_t + \gamma V(S_{t+1}) - V(S_t) \tag{2.15}$$

is the TD-error. We see that (2.14) is also of the form (2.12).

Note that if we allow the step size to vary in time, such that α_t satisfies

$$\sum_t \alpha_t = \infty, \quad \sum_t \alpha_t^2 < \infty, \tag{2.16}$$

then we have convergence to V_π (see the Robbins–Munro stochastic approximation [156] and [105]). However, it is worth noting that in the control problem, since we have non-stationary data, a constant step size is often used.

Therefore TD(0) can update after each simulated step and work in an infinite horizon setting. However, whilst the true TD-target $R_{t+1} + \gamma V_\pi(S_{t+1})$ is unbiased, bootstrapping/ using the value of $V(S_{t+1})$ will give a biased target (due to the inaccuracies of $V(S_{t+1})$). Although the variance is lower in the case of a TD-estimate compared with the MC estimate (see [173]), there is a bias-variance trade-off. TD methods exploit the Markov property of the MDP whereas MC methods look to the end and capture more path-dependent structure, therefore one method may be preferred over the other based on whether we have a fully observed environment.

A natural question to ask is, if our actions do not carry an immediate effect, is it better to simulate multiple steps? This leads to n-step TD methods. Let us define

$$G_t^{(n)} = R_t + \gamma R_{t+1} + ... + \gamma^{n-1} R_{t+n-1} + \gamma^n V(S_{t+n}).$$

Then $G_t^{(n)}$ forms the target for estimating $V(S_t)$ in the update step (2.12) of an n-step TD method.

Another method, TD(λ), uses a weighted combination of all n-step returns,

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)}, \tag{2.17}$$

with $\lambda \in [0, 1)$. In the case that $\lambda = 0$, the only non-zero term in the summation is when $n = 1$, which is just the TD(0) method (as we may have foreshadowed from its name). The weights sum to one. If we reach termination at T, then we distribute all remaining weight $1 - (1-\lambda)\sum_{n=0}^{T-t-2} \lambda^n = \lambda^{T-t-1}$ in the final step. In this case:

$$G_t^\lambda = (1-\lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} G_t^{(n)} + \lambda^{T-t-1} G_t. \tag{2.18}$$

From (2.18) we see that if $\lambda = 1$, this gives us the Monte Carlo target of G_t. If we want to have faster updates, then we can take a truncated version, for more details see [173].

This is known as the forward view TD(λ). The backward view TD(λ) involves eligibility traces. These are indications of frequency and recency of visits to the state of interest. The eligibility trace is defined to be

$$z_0(s) = 0, \quad z_t(s) = \gamma\lambda z_{t-1}(s) + \mathbb{1}_{S_t = s}. \tag{2.19}$$

We keep an eligibility trace for each state and update the state values by

$$V(S_t) \leftarrow V(S_t) + \alpha \delta_t z_t(S_t),$$

where δ_t is the TD-error (2.15). The forward and backward views are equivalent, although application of eligibility traces can be used in the Monte Carlo case too, see [173].

2.1.3 Model-free methods – control problem

Before we delve into the control problem, let us briefly discuss a classic issue in reinforcement learning problems – the exploration vs exploitation trade-off. With stochastic environments, there is a danger of getting stuck in a local minimum. Imagine you are playing slot machines, as in the context of multi-armed bandit problem. You can spend one unit to play any of n machines and the reward of each machine follows some unknown probability distribution. How should you determine the optimal strategy to win the maximum amount of money? Some intuition says that all machines should be tried at least once. But what happens if the machine with the highest expectation gives a rather low outcome the first time you play on it? If this machine is never tried again, then the fact that it will result in a greater total outcome in the long time limit will never be discovered. However if you continuously engage with machines that did not give out the best rewards, then you are likely trying sub-optimal machines and discovering lower rewards. The balance between using the best performing machine so far, and relatively unexplored machines is an example of the exploration-exploitation problem.

Greedy methods will always pick the machines which are currently believed to be the best, making it sensitive to initial performance of machines. In order to introduce exploration, the ϵ-greedy method chooses the greedy action with a probability of $1 - \epsilon$. There is an ϵ chance that the action will be chosen at random from all possible actions (including the one currently thought to be optimal). There are other strategies used, like Boltzmann exploration [38], but ϵ-greedy is by far the most popular in applied reinforcement learning.

The control problem is still formed of two parts, policy evaluation and policy improvement. Just like with the generalised policy iteration (Appendix A), it is not necessary to wait until the action values have converged before making a policy improvement. Most standard methods have an update rule of the form given in (2.12) for policy evaluation. The policy improvement is implicitly given by an ϵ-greedy method with respect to the action values.

The reinforcement learning community describes algorithms for the control problem as *on-policy* or *off-policy*. SARSA is a well-known example of an on-policy method whilst Q-learning is a prominent example of an off-policy method. On-policy methods learn the action values for an exploratory policy and aims to find a near-optimal policy. Off-policy methods use two different policies. The policy that we want to learn about is called the target policy, and the policy which generates data and ensures adequate exploration is called the behaviour policy.

We look at methods where the samples are generated by following an ϵ-greedy method to ensure adequate exploration. Using the action values as opposed to state values forms a model-free formulation. This is because we can choose the greedy policy at a state $\pi(s) = \text{argmax}_a Q(s,a)$, whereas the derived policy from state values $\pi(s) = \text{argmax}_a\{r(s,a) + \sum_{s'} p(s'|s,a)V(s')\}$ would need a model for the transition probabilities. Note that the greedy policy of choosing the action that gives the optimal action values (2.8) would result in an optimal policy if we have true action values. We can estimate the Q-values as we have done for the V-values above with MC methods and TD methods. We will work with Q-values for the rest of this section.

It can be shown that if we apply policy iteration (Appendix A) with ϵ-greedy exploration, then the updated ϵ-greedy policy, π', is an improvement on the original policy, π, in that $V_{\pi'}(s) \geq V_\pi(s), \ \forall s$ [173]. Note that if there are infinitely many/continuous actions then methods described in this section will not work. See Section 2.1.5 on solving these types of problems using policy gradients, as well as [114]. Other methods can be found in [147, 180].

Let us start with on-policy methods. We want to use an exploratory policy to generate episodes, and make improvements to this policy such that it tends towards the optimal policy. Methods which satisfy this condition and visit all states infinitely often are called GLIE (Greedy in the Limit with Infinite Exploration). An example of a GLIE method is an ϵ-greedy method with $\epsilon = 1/k$, where k is the iteration number.

For Monte Carlo methods, we still use the return G_t as an unbiased target for $Q(S_t, A_t)$ in (2.12). The same advantages of TD methods over MC methods apply also for the control problem. The one-step temporal difference method using action values is called SARSA, so named because we learn from a sequence of $\{(State, Action), Reward, (State, Action)\}$. SARSA methods use $R_t + \gamma Q(S_{t+1}, A_{t+1})$ as the target. The details can be seen in Algorithm 1. Note that in the continuing (infinite horizon) case, we would not have the outer loop over episodes.

Algorithm 1: SARSA

Parameters: step size $\alpha \in (0, 1]$, $0 < \epsilon \ll 1$.
Initialise $Q(s, a)$ for all non-terminal states $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ arbitrarily provided $Q(terminal, \cdot) = 0$
for *each episode* **do**
  Initialise starting state S
  Choose action A from S using policy derived from Q (e.g. ϵ-greedy);
  for *each step of the episode* **do**
    Take action A, observe R, S'.
    Choose A' from S' using policy derived from Q (e.g. ϵ-greedy).
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A') - Q(S, A)]$,
    $S \leftarrow S'$; $A \leftarrow A'$.
  end
end

The step size is allowed to change as a function of the timestep $\alpha = \alpha(t)$. Indeed if the step size decreases such that (2.16) is satisfied, then SARSA is shown to be convergent [167]. SARSA is an on-policy method since the action used in bootstrapping, A', is chosen from the policy we are following to generate the data.

In the same way as for state values, we can define an action value equivalent of the n-step TD methods and the TD(λ) methods. These are known as n-step SARSA and SARSA(λ); their targets for the updates are

$$Q_t^{(n)} = \sum_{k=t}^{n-1} \gamma^{i-t} R_k + \gamma^n Q(S_{t+n}, A_{t+n}),$$

and

$$Q_t^{\lambda} = (1 - \lambda) \sum_{k=1}^{\infty} \lambda^{k-1} Q_t^{(n)}$$

respectively. The backward view of SARSA(λ) with eligibility trace is entirely analogous to TD(λ) [173].

Next, let us discuss off-policy learning, which has the advantage of using data generated by one policy to potentially learn about multiple other policies. Samples generated from older policies can still be reused, making off-policy methods more data efficient. An algorithm may use data generated by humans to learn an optimal policy. Typically off-policy

learning involves an exploratory behaviour policy to learn the optimal target policy. We require the target policy π to be absolutely continuous with respect to the behaviour policy μ, $\pi \ll \mu$, that is

$$\pi(a|s) \neq 0 \implies \mu(a|s) \neq 0. \tag{2.20}$$

This means that the behaviour policy needs to be able to sample all things that the target policy would.

Importance sampling [77] is a trick often used in Monte Carlo methods. It is particularly useful where the chance of sampling in a particular region, $\mathcal{D}$, may be small and can be missed entirely if Monte Carlo simulations are run on the original distribution P. Importance sampling involves inflating the probability of sampling from $\mathcal{D}$, so that we can still examine the behaviour in this region, then re-adjusting based on the fact that an alternative sampling distribution Q was used

$$E^P[f(X)] = E^Q\left[\frac{P(X)}{Q(X)}f(X)\right],$$

where $P(\cdot)$, $Q(\cdot)$ are the probability density functions. In our case, the absolute continuity condition on the two distributions (2.20) ensures this is well-defined.

We can use this in the context of off-policy methods. If we want to learn about π whilst following μ, then for Monte Carlo methods we have

$$E^\pi[G_t] = E^\mu[\rho_t^{\pi/\mu} G_t],$$

where

$$\rho_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)}\frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})}\cdots\frac{\pi(A_T|S_T)}{\mu(A_T|S_T)}.$$

We can use $\rho_t^{\pi/\mu} G_t$ as our target in the update step of the algorithm for estimating $V(S_t)$. For the action values $Q(S_t, A_t)$ however, we would have the target $\rho_{t+1}^{\pi/\mu} G_t$ instead since our first action is fixed. Many variations based on this idea can be found in [173] and references therein.

In both cases, the target is formed of multiple products and the variance is high. Once again, temporal difference methods can reduce the number of products and hence the variance.

There is a special case of a one-step off-policy TD method for $Q(S_t, A_t)$ where no importance sampling is required. This method is known as Q-learning [188]. It uses an ϵ-greedy behaviour policy to learn a greedy target policy and updates values by

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha(R_t + \max_a Q(S_{t+1}, a) - Q(S_t, A_t)),$$

where the target is just a sample from the Bellman optimality equation (2.11). See the full algorithm in Algorithm 2 (no outer loop for the infinite horizon case).

Although the next action is chosen from an ϵ-greedy policy, we never use $Q(S_{t+1}, A_{t+1})$ for bootstrapping hence importance sampling is not needed. The n-step methods use importance sampling as detailed above and is similar to the algorithms we have already discussed [173].

2.1.4 Approximate methods

Often the size of the state space and the state-action space is too big to store precise values in a look-up table as seen in the methods described above. Keeping track of a large

Algorithm 2: Q-Learning

Parameters: step size $\alpha \in (0, 1]$, $0 < \epsilon << 1$.
Initialise $Q(s, a)$ for all non-terminal states $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$ arbitrarily provided $Q(terminal, \cdot) = 0$.
for *each episode* **do**
  Initialise state S.
  for *each step of the episode* **do**
    Choose A from S using policy derived from Q (e.g. ϵ-greedy).
    Take action A, observe R, S'.
    $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$,
    $S \leftarrow S'$.
  end
end

number of states or state-action pairs is expensive in terms of memory and convergence of prediction/control methods would be slow. We instead fit functions parametrised by some w such that

$$\hat{V}(s, w) \approx V_\pi(s), \quad \hat{Q}(s, a, w) \approx Q_\pi(s, a).$$

Intuitively speaking, function approximators suppose that the states of our problem reside in a lower dimensional subspace/manifold. Our state state $s \in \mathbb{R}^p$ can be represented in a lower dimensional feature space parametrised by $w \in \mathbb{R}^q$ such that $q \ll p$. Let x be a mapping such that $\mathbb{R}^p \to \mathbb{R}^q$, then $x(s)$ is said to be a *feature vector*.

Function approximators can be linear or nonlinear. Linear function approximators typically take the form of

$$\hat{V}(s, w) = x(s)^T w. \tag{2.21}$$

Choosing the right feature representation is important and dependent upon the problem. Popular choices of linear feature extraction methods include radial basis functions and tile coding. Neural networks are an example of nonlinear function approximators.

We want to develop methods to update the parameters w based on the data we have in order to generalise to unseen states. These function approximations can then be used in the place of exact values in algorithms. If we are solving the control problem then we also need methods that would work for non-stationary, not identically distributed data since there will be changes in policies from which the data are generated.

Our chosen parameters should optimise an objective, which is related to the error of our approximation. Let $J(w)$ be our objective function, which for example could be the mean square error (MSE) in the estimation of the state values

$$J(w) = E[(V_\pi(s) - \hat{V}(s, w))^2].$$

This objective function, which should be minimised over the parameter space, is known as a loss function in the neural networks literature.

Gradient descent finds a local minimum by repeatedly taking steps towards the descent direction of the objective function

$$w_{k+1} = w_k - \alpha \nabla_{w_k} J(w_k).$$

Stochastic gradient descent (SGD) uses a sample in the place of the expectation in the objective function. The update rule for SGD on the MSE is

$$w_{k+1} = w_k - 2\alpha(V_\pi(s) - \hat{V}(s, w_k))\nabla_{w_k}\hat{V}(s, w_k).$$

The simplicity of finding the derivative $\nabla_{w_k}\hat{V}(s, w_k)$ is what makes linear function approximation methods and neural networks so widely used. The gradient for linear function approximators is just the feature vector $x(s)$. For neural networks we can find the gradients efficiently through backpropagation (reverse mode of automatic differentiation) [17].

Using linear function approximators in combination with an MSE objective means the objective function is quadratic (and thus convex) in the parameters w. In this case SGD theoretically finds the global optimum.

However, unlike with supervised learning, we do not have true values of V_π or Q_π to be used in any objective functions, therefore we substitute in a target value. These targets are precisely the targets we met previously. That is, for estimating the state values: the Monte Carlo target is G_t, for TD(0), the target is the TD-target (2.14) and for TD(λ), it is G_t^λ (2.17). The only difference is in the backwards view, the eligibility trace for function approximators is

$$z_0(s) = 0, \quad z_t(s) = \gamma\lambda z_{t-1}(s) + \nabla_{w_t}\hat{V}(s, w_t). \tag{2.22}$$

When we consider that look-up tables can be thought of as a special case of linear function approximator where the features are indicator functions of whether we are currently in that specific state, then this definition coincides with (2.19).

For the action values, the targets are entirely analogous to the table representation, bar the amendment to the eligibility trace: $\nabla_{w_t}\hat{Q}(s, a, w_t)$ instead of $\mathbb{1}_{S_t=s,A_t=a}$. Similar to supervised learning, these targets take the place of the true value and we can update the parameters w to reduce the objective function $J(w)$ and learn a (locally) optimal representation.

By approximating our state or action values with a function approximator, this would solve the prediction problem. To solve the control problem, we still use action value approximations since it is model-free. We evaluate the policy for $\hat{Q}(s, a, w)$ as discussed and, when the number of actions is small, we can improve our policy for example by taking an ϵ-greedy policy with respect to $\hat{Q}(a, s, w)$.

The methodology discussed above can be used for online learning where the parameters are updated with each new sample. However, this is not sample efficient. If obtaining data is expensive or we only have finite amount of data, we need batch methods where the same data are used (repeatedly) to adjust the parameters.

It is worth noting that the combination of function approximation, bootstrapping and off-policy learning results in unstable algorithms (the value function diverges) with examples as seen in [190] and [178]. Sutton and Barto coined the term 'The deadly triad' to describe this phenomenon [173]. Therefore we cannot naïvely perform methods when we have such a combination in our problem. This is still an area of active research.

There have been numerous developments in deep reinforcement learning to address the stability issue; most rely on implementation heuristics. This area was popularised by [123]. The method introduced in [123] is known as Deep Q-Networks. The policy improvement is based on Q-learning. The behaviour policy is an ϵ-greedy policy with respect to the estimated action values $\hat{Q}(s, a, w)$ (note that the state s has gone through a stage of preprocessing); the target is $y = r + \text{argmax}_{a'}\hat{Q}(s', a', \tilde{w})$. This paper utilises two ideas to try and stabilise the method: *experience replay* and *fixed Q-targets*.

Experience replay stores samples of (S_t, A_t, R_t, S_{t+1}) pooled from many complete episodes into a replay memory $\mathcal{D}$. The training is done by performing Q-learning on samples drawn uniformly from the replay memory. This way, the data will be decorrelated.

One of the main reasons why function approximation with bootstrapping is unstable for the control problem is because the target in the bootstrapped case depends on the parameters w too. Therefore as these parameters are adjusted, the target will also move. To reduce the effect of this problem, the paper [123] stores two sets of weights, w and $\tilde{w}$, which are updated at different timescales. Every C steps, the weights w is copied to $\tilde{w}$. During the next C iterations, w is updated by SGD as discussed to minimise residual sum of squares (RSS) between the target, $y = r + \text{argmax}_{a'}\hat{Q}(s', a', \tilde{w})$ and $\hat{Q}(s, a, w)$. Note that it is $\tilde{w}$ which is used to compute the target in these C steps. These parameters, $\tilde{w}$, are updated at the end of this cycle and the process repeats. Letting the parameters update towards the target for a period of time before changing the target is observed to give more stable results in practice (intuitively this is similar to convergence of the value function in policy iteration – described in Appendix A). Batch normalisation [92] became popular after Deep Q-Networks and can be used in conjunction.

Since then, there have been numerous other publications in the area. In particular a method for large discrete action spaces is discussed in [64]. Here the authors consider the case where the action may have been a continuous variable that was discretised. Their idea involves mapping the action proposed (which can be in continuous space and not a valid action) into a set of the approximate k-nearest neighbour, $\mathcal{A}_k$ (feasible actions), and then choosing the action

$$a = \text{argmax}_{a_j \in \mathcal{A}_k} Q(s, a_j, w).$$

This is known as the Wolpertinger policy algorithm. The disadvantage of this is that it involves looking up the k-nearest neighbour (in L_2) in the set of feasible actions $\mathcal{A}$, although an approximate lookup can be done in logarithmic time [64].

2.1.5 Policy gradients

The methods in the previous section where we learn the value function and derive an implicit policy from it are known as *value-based* methods. In this section we look at *policy-based* methods where we do not use value functions but instead learn an explicit policy. We approximated V_π and Q_π with a neural network previously, however there is no reason we cannot approximate the policy itself

$$\pi(a|s) \approx \pi(a|s, \theta) = P(A_t = a|S_t = s, \theta_t = \theta).$$

We may denote $\pi(a|s, \theta)$ as π_θ for notational simplicity. These are also sometimes referred to as *actor-only* methods and the value-based methods may be called *critic-only* methods.

The advantages of such methods are that it is applicable to large or continuous action spaces and stochastic policies can be learnt. Although stochastic policies may feel less intuitive, a simple example of an optimal policy that is stochastic is the policy of rock-paper-scissors. The optimal policy is to pick any action with probability 1/3. This is a 2-player, 3-action game, with a mixed-strategy equilibrium. This type of game is explored further in Chapter 8.

Stochastic policies are particularly useful when our system can only be partially observed. Learning a stochastic policy is easy and natural with policy-based methods compared with value-based methods.

As with value-based methods, we can extract features $\phi(s,a)$ from our state so that for some function g,

$$\pi(a|s,\theta) = g(\phi(s,a),\theta).$$

Note here we require our features to be a function of the action a. These features may be selected based on prior knowledge about the system. If we expect the policy to be simpler to parametrise compared with the value function, then using a policy-based method can be more effective. This problem-specific issue should be given prior thought before choosing between a policy-based or value-based approach.

The question is how can we assess the quality of an approximate policy; what should our objective function be? To solve the control problem we should be finding an optimal solution, getting the maximum expected reward from the system. Its performance measure should be a function of the reward received following it. A common objective function for episodic (finite horizon) environments is

$$J(\theta) = V_{\pi_\theta}(s_1),$$

the estimated state value from the starting state. For the continuing (infinite horizon) case, we suppose the process is ergodic and that the stationary distribution of states under policy π_θ is μ, then we can use the expected reward

$$\begin{aligned} J(\theta) &= \lim_{t\to\infty} E[R_t|S_1, A_1, ..., A_t \sim \pi_\theta] \\ &= \sum_s \mu(s) \sum_a \pi(a|s,\theta) \sum_{s',r} p(s', r|s, a)r, \end{aligned}$$

as a performance measure. From here on, continuous optimisation methods such as derivative-free optimisation or gradient descent can be applied to maximise the objective function $J(\theta)$. We refer to $\nabla_\theta J(\theta)$ as the *policy gradient*. Methods which update the parameters by

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta \hat{J}(\theta), \tag{2.23}$$

where $\nabla_\theta \hat{J}(\theta)$ is some stochastic estimate with expectation relating to $\nabla_\theta J(\theta)$, are known as *policy gradient methods*.

The policy gradient can be approximated numerically by e.g. finite differences [107]. If we suppose that the policy is differentiable when non-zero and we know the form that $\nabla_\theta \pi(a|s,\theta)$ takes, then we can also study the policy gradient analytically. One thing to note is that the policy parameters would affect the actions, which in turn would affect the distribution of states seen. The policy gradient theorem gives an analytic form that does not involve state distributions by utilising action values.

Theorem 2.1.5. *The policy gradient theorem states that*

$$\nabla_\theta J(\theta) = K \sum_s \mu(s) \sum_a Q_{\pi_\theta}(s,a) \nabla_\theta \pi(a|s,\theta) \tag{2.24}$$

$$= K E_{\pi_\theta} \left[\sum_a Q_{\pi_\theta}(S_t, a) \nabla_\theta \pi(a|S_t,\theta) \right], \tag{2.25}$$

where μ is the stationary distribution of the states under policy π, and the constant K is equal to the expected length of an episode for the finite horizon case or $K = 1$ in the infinite horizon case [173].

Note that the constant value can be absorbed into the step size parameter when we use gradient descent and therefore is often omitted.

We can rewrite (2.25) as

$$\begin{aligned}\nabla_\theta J(\theta) &= KE_{\pi_\theta}\left[\sum_a \pi(a|S_t,\theta)Q_{\pi_\theta}(S_t,a)\frac{\nabla_\theta\pi(a|S_t,\theta)}{\pi(a|S_t,\theta)}\right] \\ &= KE_{\pi_\theta}\left[Q_{\pi_\theta}(S_t,A_t)\frac{\nabla_\theta\pi(A_t|S_t,\theta)}{\pi(A_t|S_t,\theta)}\right] \\ &= KE_{\pi_\theta}\left[Q_{\pi_\theta}(S_t,A_t)\nabla_\theta \log\pi(A_t|S_t,\theta)\right], \end{aligned} \tag{2.26}$$

where $\nabla_\theta \log\pi(A_t|S_t,\theta)$ is referred to as the score (or score function) in the statistics literature or eligibility vector in [173].

Suppose we make a suitable choice of function approximation such that the eligibility vector is known, we can apply stochastic gradient descent and use the same targets for $Q_{\pi_\theta}(S_t,A_t)$. However, for policy-based methods we do not have value function estimates so we cannot do the bootstrapping needed for temporal difference methods. For Monte Carlo policy gradient methods, we can still use G_t as an unbiased estimator for $Q_{\pi_\theta}(S_t,A_t)$ in (2.26) and use this to update the parameters with (2.23).

We can reduce the variance by introducing a baseline function $b(s)$. We first note that

$$\sum_a b(s)\nabla_\theta\pi(a|s,\theta) = b(s)\nabla_\theta\sum_a \pi(a|s,\theta) = b(s)\nabla_\theta 1 = 0,$$

therefore the policy gradient remains unchanged if we subtract this term from (2.24)

$$\begin{aligned}\nabla_\theta J(\theta) &= K\sum_s \mu(s)\sum_a \left(Q_{\pi_\theta}(s,a) - b(s)\right)\nabla_\theta\pi(a|s,\theta) \\ &= KE_{\pi_\theta}\left[(Q_{\pi_\theta}(S_t,A_t) - b(S_t))\nabla_\theta\log\pi(A_t|S_t,\theta)\right].\end{aligned}$$

A commonly used baseline function is the state values $V_{\pi_\theta}(s)$. The intuition behind this is that the state value captures how valuable it is to be in this state taking into account possible future states. Subtracting this away from the action values captures any advantageous effect of taking the specific action a over any other actions whilst in state s. The difference

$$A_{\pi_\theta}(s,a) = Q_{\pi_\theta}(s,a) - V_{\pi_\theta}(s) \tag{2.27}$$

is known as the advantage function. Computing this function requires that the value functions are estimated somehow. With the REINFORCE method [173], the state values are learnt with MC methods.

To summarise, policy gradient methods approximate the policy gradient using an estimate $\hat{Q}$ of $Q_{\pi_\theta}(s,a)$ through Monte Carlo methods. A baseline may be used as a variance reduction technique. The parameters can then be updated by stochastic gradient descent

$$\theta_{t+1} = \theta_t + \alpha(\hat{Q}(s,a) - b(s))\nabla_\theta\log\pi(a|s,\theta). \tag{2.28}$$

2.1.6 Actor-critic methods

As before, learning can be slow with Monte Carlo Policy gradients. This motivates us to maintain a parametrised value function to bootstrap from so that we can apply temporal

difference methods. Actor-critic methods combine value-based and policy-based methods – both a value function and policy function are learnt. Therefore actor-critic methods maintain two sets of parameters, one for the actor

$$\pi(a|s,\theta) \approx \pi(a|s),$$

and one for the critic

$$\hat{V}(s,w) \approx V_\pi(s).$$

The critic solves the policy evaluation part of the problem with a Monte Carlo or temporal difference method. Note that the REINFORCE method mentioned above does not count as an actor-critic method since the estimated state values are only used for the baseline function, and not for bootstrapping.

We would like to use the advantage function (2.27) to reduce variance. It may seem at first that we would need estimates of both V and Q in order to estimate the advantage function. In fact the TD-error (2.15) is an unbiased estimate of the advantage function

$$\begin{aligned} E_{\pi_\theta}[\delta_t^{\pi_\theta}|S_t=s,A_t=a] &= E_{\pi_\theta}[R_t+\gamma V_{\pi_\theta}(S_{t+1})-V_{\pi_\theta}(S_t)|S_t=s,A_t=a] \\ &= Q_{\pi_\theta}(s,a)-V_{\pi_\theta}(s) \\ &= A_{\pi_\theta}(s,a). \end{aligned}$$

Intuitively the TD-error is the difference between how well we think we are doing compared with another evaluation of the situation after one action. The difference in assessment is likely due to the action that we chose, which is exactly the same intuition behind the advantage function. Thus we only need to estimate the state values, which reduces the dimension of the problem – the action values are not required.

Hence in actor-critic methods we use the function approximation $\hat{V}_{\pi_\theta}(s,w)$ instead of the true state values to approximate the TD-error (2.15). We denote this approximate TD-error as $\hat{\delta}_t^{\pi_\theta}$. We can then update the actor parameters with $\hat{\delta}_t^{\pi_\theta}$ in the place of the advantage function in (2.28). Other temporal difference actor-critic methods may use eligibility traces, see [173] for examples. For a review of actor-critic methods pre-2012 see [81].

In recent times, actor-critic methods with neural network approximations have gained popularity. Notable algorithms include A3C [122], DDPG [114], and PPO [163]. Some of these (e.g. [122]) are developments which focused more on the efficient implementation of deep reinforcement learning methods through parallel computing.

Recall that value-based methods learn deterministic policies by estimating action values. Taking the argmax of the action-values becomes problematic with large or continuous action spaces. The *Deep Deterministic Policy Gradient* (DDPG) method [114] is built on deterministic policy gradients [165]. We have discussed stochastic policy gradients in the section above, which are used to update the parameters of the function approximator when the behaviour policy is stochastic, $\pi_\theta(a|s)$. How does this change when we want to use deterministic policies, $\pi_\theta(s)$, for the behaviour policy instead? In [165] it is shown that

$$\nabla_\theta J(\theta) = E_\pi\left[\nabla_\theta \pi(S_t,\theta)\nabla_a Q_{\pi_\theta}(S_t,a)|a=\pi(S_t,\theta)\right]$$

is the policy gradient in this case. In addition, this is the limit of the stochastic policy gradient as defined in (2.24) when the policy variance tends to zero (under certain conditions) [165]. Since the expectation is no longer over the possible actions from each state, less samples would be required when applying reinforcement learning methods, potentially speeding up convergence. The DDPG method makes use of this to update the network parameters.

For the algorithm as given in [114] see Algorithm 3. We note in particular that to ensure exploration, noise is generated from some process and added onto the action. Similar to the fixed Q-targets for Deep Q-Networks, there are two sets of parameters for the critic and two sets for the actor. This is to ensure that the targets do not change too rapidly which may cause instability. The weights used for the targets are updated on a much slower scale with $\tau \ll 1$. Note that any optimiser, for example Adam [106], can be used.

Algorithm 3: DDPG Algorithm

Randomly initialise critic network $Q(s, a|\theta^Q)$ and actor $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ.
Initialise target network Q' and μ' with weights $\theta^{Q'} \leftarrow \theta^Q, \theta^{\mu'} \leftarrow \theta^\mu$.
Initialise replay buffer R.
for *episode = 1, M* **do**
  Initialise a random process $\mathcal{N}$ for action exploration.
  Receive initial observation state s_1.
  for $t = 1, T$ **do**
    Select action $a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t$ according to the current policy and exploration noise.
    Execute action a_t and observe reward r_t and new state s_{t+1}.
    Store transition (s_t, a_t, r_t, s_{t+1}) in R.
    Sample a random minibatch of N transitions (s_i, a_i, r_i, s_{i+1}) from R.
    Set $y_i = r_i + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^Q)$.
    Update critic by minimising the loss: $L = \frac{1}{N}\sum_i (y_i - Q(s_i, a_i|\theta^Q))^2$.
    Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta^\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s, a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}.$$

    Update the target networks:

$$\theta^{Q'} \leftarrow \tau\theta^Q + (1-\tau)\theta^{Q'},$$

$$\theta^{\mu'} \leftarrow \tau\theta^\mu + (1-\tau)\theta^{\mu'}.$$

  end
end

2.2 Path Signatures

As discussed in Section 2.1.4, intuitively, function approximation looks to represent high dimensional functions using a lower-dimensional manifold. Choosing relevant features from the original states is non-trivial. Many different feature representations have been used in reinforcement learning, e.g. radial basis functions and tile coding [173]. Another option which has not been explored specifically in the context of reinforcement learning is *path signatures*.

Path signatures were first used by Chen in 1958 [41] but have since been applied to differential equations driven by rough paths [119]. In recent times, they have been applied

to machine learning problems [43] as features of time-series like data, for example, in the Chinese handwriting competition [80]. They have also been used in optimal execution problems to maximise the wealth of an investor [97, 35]. Signatures are of particular interest since they have geometric meaning. In particular, they can capture the order of events, as well as handle regular and irregular data within the same framework. They also have algebraic properties, for example, they are invariant to path reparametrisation.

In this section we examine some basic theory, taking definitions from [43], so that we can exploit signatures as features in our model.

2.2.1 Definition and properties of the signature

Let a path be defined as a continuous mapping $X : [a, b] \to \mathbb{R}^d$. Recall that the path integral is a Riemann–Stieltjes integral written as

$$\int_a^b f(X_t)dX_t,$$

and if X_t is differentiable with respect to t then we can write this as a Riemann integral

$$\int_a^b f(X_t)dX_t = \int_a^b f(X_t)\dot{X}_t dt.$$

For each coordinate of $X = (X^1, X^2, ..., X^d)$, we have a 1-D path $X^i : [a, b] \to \mathbb{R}$, $i = 1, ..., d$.

Let us now denote

$$S(X)^i_{a,b} = \int_{a<s<b} dX^i_s = X^i_b - X^i_a, \tag{2.29}$$

and let $S(X)^{i,j}_{a,t}$ be the double iterated integral

$$S(X)^{i,j}_{a,b} = \int_{a<s<b} S(X)^i_{a,s} dX^j_s = \int_{a<r<s<b} dX^i_r dX^j_s.$$

Suppose higher order iterated integrals are similarly defined. The signature is the collection of all such iterated integrals.

Definition 2.2.1 (Signature)**.** *The signature of a path is an infinite collection:*

$$S(X)_{a,b} = (1, S(X)^1_{a,b}, ..., S(X)^d_{a,b}, S(X)^{1,1}_{a,b}, S(X)^{1,2}_{a,b}, ...).$$

The elements of the multi-index of the signature

$$W = \{(i_1, ..., i_k) | k \geq 1, i_1, ..., i_k \in \{1, ..., d\}\}$$

are referred to as *words* on the *alphabet* $A = \{1, .., d\}$.

We refer to the collection of terms with a multi-index of length k as the kth level of the signature. The signature is truncated when we use them in applications. We see from the above the number of terms when we truncate the signature at the nth level is $\sum_{i=0}^{n} d^i$.

As demonstrated above, the first level signature terms (2.29) are the increment in the ith dimension. The second order terms also have geometric interpretation, they can be thought of in terms of the Lévy area. These terms are important in that they capture the order of events— how different dimensions move together as a pair. Higher order terms are more abstract in their meaning but captures local information.

The signature is invariant with time reparametrisation, where a reparametrisation is a surjective, continuous, non-decreasing function $\phi : [a, b] \to [a, b]$, see [43]. The uniqueness of signatures for paths, up to tree-like equivalence, is proved in [83].

Another useful property is that the product of any two terms in the signature can be written as a sum of a collection of terms in the signature whose multi-index only depends on the multi-index of the original two terms, this is known as the shuffle product identity. To formalise this property, we first define the shuffle product. This product is so named because it is an order preserving shuffle. Imagine shuffling a deck of cards by riffling. The combined deck afterwards will still have the elements of each half of the deck in their respective order.

Definition 2.2.2 (The Shuffle Product)**.** *Suppose $I = (i_1, \dots, i_k)$ and $J = (j_1, \dots, j_m)$ are multi-indices (words on the alphabet) such that $i_1, \dots, i_k, j_1, \dots, j_m \in \{1, \dots, d\}$. Define*

$$(r_1, \dots, r_k, r_{k+1}, \dots r_{k+m}) = (i_1, \dots, i_k, j_1, \dots j_m).$$

Let σ be a permutation on the set $\{1, \dots, k+m\}$. Then the shuffle product is a set such that

$$I \sqcup\!\sqcup J = \{(r_{\sigma^{-1}(1)}, \dots, r_{\sigma^{-1}(k+m)}) | \sigma(1) < \dots < \sigma(k), \sigma(k+1) < \dots < \sigma(k+m)\}.$$

As an example take $A = (1, 2)$ and $B = (4, 5)$, then the shuffle product $A \sqcup\!\sqcup B = \{(1, 2, 4, 5), (1, 4, 2, 5), (1, 4, 5, 2), (4, 1, 2, 5), (4, 1, 5, 2), (4, 5, 1, 2)\}$.

With this definition we are ready to define the shuffle product identity.

Theorem 2.2.3 (The Shuffle Product Identity)**.** *For a path $X : [a, b] \to \mathbb{R}^d$, let $I = (i_1, \dots, i_k)$ and $J = (j_1, \dots, j_m)$ be multi-indices such that $i_1, \dots, i_k, j_1, \dots, j_m \in \{1, \dots, d\}$. Then we always have*

$$S(X)^I_{a,b} S(X)^J_{a,b} = \sum_{K \in I \sqcup\!\sqcup J} S(X)^K_{a,b}.$$

Another well-known property is Chen's identity. This is the property which gives the signature the description of "a homomorphism from the monoid of paths (under concatenation) into the grouplike elements of the free tensor algebra" [118]. We now make this statement more precise and concrete.

Definition 2.2.4 (Formal power series)**.** *Let $e_1, \dots, e_d$ be d formal indeterminates (used as placeholders in formal algebra). Let $A = \{1, \dots, d\}$. A (non-commuting) formal power series takes the form*

$$\sum_{k=0}^{\infty} \sum_{i_1, \dots, i_k \in A} \lambda_{i_1, .., i_k} e_{i_1} \dots e_{i_k}, \tag{2.30}$$

where second summation is over all multi-indices with elements in A and $\lambda_{i_1, .., i_k}$ are real numbers.

Note that we consider the non-commuting power series only, that is $e_1 e_2$ and $e_2 e_1$ are distinct. We can define addition and scalar multiplication on formal power series as

$$\left(\sum_{k=0}^{\infty} \sum_{i_1, \dots, i_k \in A} \lambda_{i_1, .., i_k} e_{i_1} \dots e_{i_k} \right) + \left(\sum_{k=0}^{\infty} \sum_{i_1, \dots, i_k \in A} \mu_{i_1, .., i_k} e_{i_1} \dots e_{i_k} \right)$$
$$= \sum_{k=0}^{\infty} \sum_{i_1, \dots, i_k \in A} (\lambda_{i_1, .., i_k} + \mu_{i_1, .., i_k}) e_{i_1} \dots e_{i_k},$$

and

$$c\left(\sum_{k=0}^{\infty}\sum_{i_1,\ldots,i_k\in A}\lambda_{i_1,..,i_k}e_{i_1}\ldots e_{i_k}\right)=\sum_{k=0}^{\infty}\sum_{i_1,\ldots,i_k\in A}c\lambda_{i_1,..,i_k}e_{i_1}\ldots e_{i_k}.$$

Let us define the product $\otimes$ between the indeterminates as

$$e_{i_1}\ldots e_{i_k}\otimes e_{j_1}\ldots e_{j_m}=e_{i_1}\ldots e_{i_k}e_{j_1}\ldots e_{j_m},$$

and extend this naturally to the formal power series

$$\begin{aligned}&\left(\sum_{k=0}^{\infty}\sum_{i_1,\ldots,i_k\in A}\lambda_{i_1,..,i_k}e_{i_1}\ldots e_{i_k}\right)\otimes\left(\sum_{k=0}^{\infty}\sum_{i_1,\ldots,i_k\in A}\mu_{i_1,..,i_k}e_{i_1}\ldots e_{i_k}\right)\\&=\lambda_0\mu_0+\sum_{i=1}^{d}(\lambda_0\mu_i+\lambda_i\mu_0)e_i+\sum_{i,j=1}^{d}(\lambda_0\mu_{i,j}+\lambda_i\mu_j+\lambda_{i,j}\mu_0)e_ie_j+\ldots\end{aligned}$$

The space of formal power series is an algebra when equipped with addition, scalar multiplication and the product $\otimes$. This space of formal power series is often referred to as the tensor algebra of $\mathbb{R}^d$.

The coefficients of the formal power series have been chosen to have the same indexing set as terms in the signature. We can thus express the signature of X as a formal power series

$$S(X)_{a,b}=\sum_{k=0}^{\infty}\sum_{i_1,\ldots,i_k\in A}S(X)_{a,b}^{i_1,..,i_k}e_{i_1}\ldots e_{i_k},$$

where the first term for $k=0$ is 1.

Next let us define what is meant by concatenation in the case of paths.

Definition 2.2.5 (Concatenation of paths)**.** *Suppose we have two paths in d-dimensions, $X:[a,b]\to\mathbb{R}^d$ and $Y:[b,c]\to\mathbb{R}^d$. The concatenation of the two paths $X*Y:[a,c]\to\mathbb{R}^d$ is the path for which $(X*Y)_t=X_t$ for $t\in[a,b]$ and $(X*Y)_t=X_b+(Y_t-Y_b)$ for $t\in[b,c]$.*

That is, the concatenation of the path can be thought of as 'gluing' the end of one path with the start of the next path in $\mathbb{R}^d$.

Theorem 2.2.6 (Chen's identity)**.** *Suppose we have two paths in d-dimensions, $X:[a,b]\to\mathbb{R}^d$ and $Y:[b,c]\to\mathbb{R}^d$. Then*

$$S(X*Y)_{a,c}=S(X)_{a,b}\otimes S(Y)_{b,c}.$$

Let

$$V^{\otimes n}=\underbrace{V\otimes V\otimes\cdots\otimes V}_{\text{n products}}$$

be the nth power with respect to $\otimes$. For x, a formal power series of the form given by (2.30) with $\lambda_0>0$, the logarithm is defined as

$$\log x=\log(\lambda_0)+\sum_{n\geq1}\frac{(-1)^n}{n}\left(1-\frac{x}{\lambda_0}\right)^{\otimes n}. \tag{2.31}$$

Definition 2.2.7 (Log signature). *The log signature of a path* $X : [a, b] \to \mathbb{R}^d$ *is the formal power series* $\log S_{a,b}(X)$*, with* log *as defined by (2.31).*

The log signature can be thought of as a more compact representation of the signature such that similar information is captured in far fewer terms for the same level of truncation [154]. Log signatures have been successfully used in [126] as an efficient way to reduce the dimension of the input features whilst maintaining high performance for very big datasets (timeseries of greater than 17,000 points).

2.2.2 Signatures of data and their transforms

When working with real data, despite the fact that some variables are inherently continuous, we only observe them at discrete intervals. As a result, we need to find a way to interpret new data releases as coming from some underlying continuous time series and exploit connections of the inputs to best model the target variable. The path signature is a natural feature set to consider for time-series data, and more generally high-dimensional paths. There may also be data quality issues, for example our data may be irregular, and have mixed frequency or be missing across the different variables measured. By interpreting the timeseries as a path object, all of these problems can be handled.

As mentioned in Section 1.3, we intuitively expect there to be some geometric aspect of our problem in terms of the optimal sequences of aircraft. Therefore, we anticipate that extracting path signatures from our data would give useful features for predicting the optimal policy. In this section, we go through how to handle discrete data points to extract path signatures.

To illustrate signature computation and transformations of such data stream, we use a toy example. Let a two-dimensional stream be given by:

$$\{t_i\}_{i=0}^{3} = \{0, 1, 2, 3\},$$
$$\{X_i\}_{i=0}^{3} = \{1, 5, 2, 6\}.$$

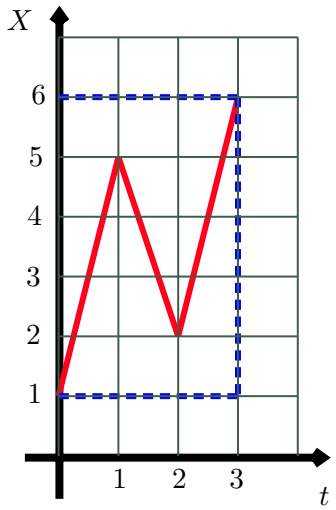


Figure 2.2: Example of a path.

An immediate question is that since a path is defined to be a continuous mapping, how can we interpolate the discrete data points? A simple method is linear interpolation between

the points but other methods like rectilinear interpolation can be used [43]. See the linear interpolated path in Figure 2.2.

The signature and the log signature of the linearly interpolated path truncated at the second level are $\{1, 3, 5, 4.5, 7.5, 7.5, 12.5\}$ and $\{3, 5, 0\}$ respectively. We see from Figure 2.2 that the second and third term in the truncated signature correspond to the increment in t and in X respectively. The fifth and sixth terms are $S(X)^{1,2}_{0,3}$ and $S(X)^{2,1}_{0,3}$, they represent the areas above and below the path respectively (both 7.5 in this case). The fourth term $S(X)^{1,1}_{0,3}$ is calculated by $\left(S(X)^{1}_{0,3}\right)^2/2 = 3^2/2 = 4.5$ and the final term $S(X)^{2,2}_{0,3}$ is given by $\left(S(X)^{2}_{0,3}\right)^2/2 = 5^2/2 = 12.5$.

We can also define transformations on the paths to give a better representation of the data and extract more useful features. The lead-lag transformation increases the dimension by having another feature path with an offset to the original features. This is best illustrated with an example. The lead-lag transformation of X is

$$LeadLag : X = \{1, 5, 2, 6\} \rightarrow \begin{cases} X^{lead} = \{1, 5, 5, 2, 2, 6, 6\}, \\ X^{lag} = \{1, 1, 5, 5, 2, 2, 6\}. \end{cases}$$

The plot of this transform can be seen in Figure 2.3.

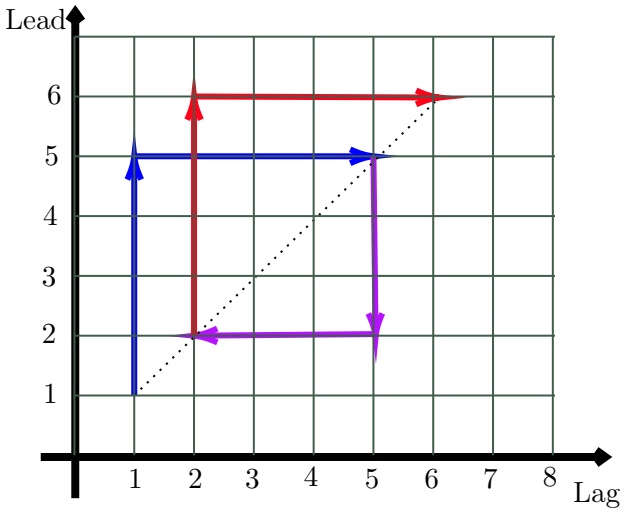


Figure 2.3: Example of a lead-lag transformation.

The advantage of the lead-lag transform is that the area of the transformed path is related to the quadratic variation of the original path [43].

We can also find the cumulative sums of the features

$$CS(\{X\}) = \left\{ X_1, X_1 + X_2, \ldots, \sum_{i=1}^{n} X_i \right\}.$$

For example the cumulative sum of X is $\{1, 6, 8, 14\}$. This may be useful since the truncated signature of the lead-lag transformation for the cumulative sums is related to the statistical moments, see [43] for more details.

The use of signatures as features in machine learning problems has gained a lot of popularity, giving state-of-the-art performance in applications including early detection of

sepsis [127], malware detection [47], and time series generation [135]. Signature kernels have also been developed [159, 112].

In addition, differential equation-inspired neural networks have become increasingly popular [42]. In this context, signatures have been used in Neural Rough differential equations [126]. Motivated by controlled differential equations, neural controlled differential equations (Neural CDEs)[104] can be thought of as a continuous-time extension of recurrent neural networks (RNNs). This also requires the construction of a continuous control path from discrete observations. For online prediction tasks, the underlying control path should be causal [125]. This modification still achieves state-of-the-art performance on prediction problems. Details of further work completed over this DPhil on neural CDEs and other signature applications can be found in Appendix J.

2.3 Multigrid methods

As we will use discrete models in this thesis, an interesting mathematical problem that often arises in discrete models is how to take discretisations of increasingly finer resolution. Continuous PDE problems often cannot be solved analytically. This has prompted the development of a wide range of numerical methods to solve the discretised versions, e.g. finite difference or finite elements. These methods transform the problem from solving differential equations to finding the solution of linear equations.

Solving $Ax = b$ efficiently has been at the core of numerical linear algebra research, and remains relevant today. Numerical methods [33] can be broadly split into two categories: direct methods and iterative methods. Whilst the divide is not strictly separated by mathematical formulae, direct methods are generally those that can solve the problem within a fixed number of iterations. Usually it is not possible to obtain a meaningful answer if we stop a direct method half-way through. Gaussian elimination is an example of a direct method.

One major problem with direct methods is storing large matrices in memory, which can be computationally prohibitive. Linear equations arising from discretisation of PDE problems are often sparse. We can exploit this structure with iterative methods. These methods start with an initial estimate and aim to get incrementally closer to the true solution.

Well-known traditional iterative methods include Jacobi and Gauss–Seidel. Suppose we discretise our problem into N intervals. When performing local mode analysis on these methods [30, 195], we find that $\mathcal{O}(N^2)$ iterations are required for the convergence of low frequency modes [171]. As the dimensions increase, these methods become infeasible. Multigrid methods can solve sparse systems to high accuracy in a fixed number of iterations [171]. This is because the low frequency modes appear as higher frequency ones on the coarse grid. For example, two complete periods in $N + 1$ discrete points on the fine grid becomes two complete periods on $N/2 + 1$ points on a grid with double the mesh size. Although the initial motivation of multigrid methods was to solve elliptic PDEs on simple domains, the theory has advanced to generalise to many problems including nonlinear problems on complex/unstructured domains [63] and singular perturbation problems [74].

This class of method from PDEs has inspired work in machine learning to design neural networks that would learn information from different orders (e.g. [157, 176]).

2.3.1 Concept of multigrid methods

In this section, we give a brief overview of multigrid methods, see [30], [82], and [189] for more details. An intuitive explanation for the efficiency of multigrid methods can be found in [195]. Multigrid methods can be described as multiscale, multilevel or multi-resolution in the literature.

Multigrid methods were first investigated by Fedorenko [68, 69]. The original application of these methods were mainly simple boundary value problems. A lot of the earlier work on multigrid methods was intuition-based and applied to only quasi-uniform meshes [82]. Since its introduction, numerous convergence properties for general grids have been proved, see for example [197].

As mentioned above, for coarser grids the lower frequency modes become higher frequency and thus can be reduced rapidly by methods such as Jacobi. These methods are called smoothing method/smoothers and form an essential part of the multigrid methodology. A popular smoothing method is the damped Jacobi method, this is an adaptation of the Jacobi method with some *damping factor*, $\omega \in \mathbb{R}$. More precisely, suppose at the kth iteration, the update computed by Jacobi is $u^{(J)}$, then the damped Jacobi update is given by

$$u^{(k)} = (1-\omega)u^{(k-1)} + \omega u^{(J)}.$$

The damping factor, ω, describes the weighted average between the new Jacobi solution and the current estimate to give the next estimate. Note how this parallels with the step size parameter in machine learning methods.

Intuitively, we want to transfer our problem from a fine grid to a coarse grid to tackle low frequency errors, this is known as *coarse grid correction*. Since the dimension of the problem is smaller on the coarse grid, this will be computationally efficient. However, we cannot naïvely change the grid size and then apply a smoothing method. It is important to realise that by transferring to a rough grid, high frequency information is lost. Completely distinct functions on a fine grid can appear identical on coarse grids, this is known as *aliasing*. Therefore, it is important to extract information from the fine grid in conjunction with the coarse grid(s).

As its name suggests, multiple grid sizes will be used in multigrid methods. Therefore the core ingredients of multigrid methods include operators to transfer the problem between these grids. The *restriction operator* R transfers vectors from a fine grid to a coarser grid whilst the *interpolation operator* I performs the reverse. The interpolation process is also known as *prolongation*. The grid sizes often change in factors of two so that the grids are nested.

In Figure 2.4, we see the two-level V-Cycle method, so named because of its shape. Many multigrid methods uses the steps seen here. Let us go though the key processes in this method. Suppose we are solving a simple boundary value problem.

Let h be the mesh size of the fine grid and $2h$ be the size of the coarse grid. Suppose we have discretised our problem on the fine grid using e.g. finite differences to obtain linear equations

$$A^{(h)}u^{(h)} = f^{(h)}.$$

Given an initial guess $\tilde{u}^{(h)}$ for $u^{(h)}$, we first apply a number of steps of an iterative strategy like the damped Jacobi to update $\tilde{u}^{(h)}$. This step is known as pre-smoothing. We compute the residual of the current solution

$$r^{(h)} = f^{(h)} - A^{(h)}\tilde{u}^{(h)},$$

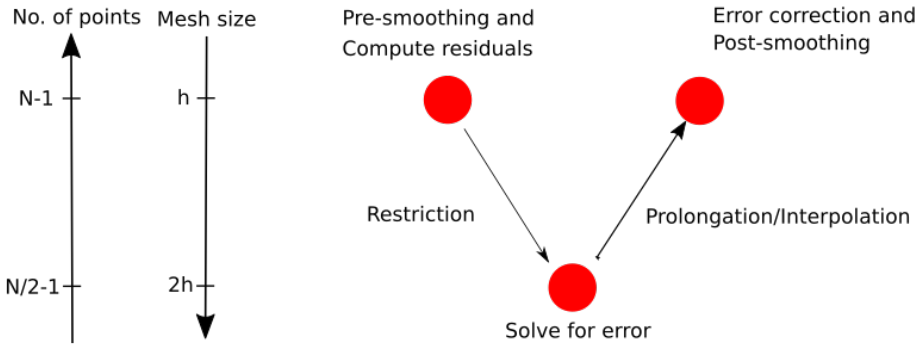


Figure 2.4: A two grid V-cycle with labelled processes.

and then restrict the residual onto the coarse grid

$$r^{(2h)} = Rr^{(h)}. \tag{2.32}$$

Suppose our points are indexed by 0. Common choices for the restriction operator R are *injection*

$$(Rv)_i = v_{2i}, \quad i = 0, \dots \frac{N}{2} - 1,$$

where the corresponding points on the fine grid is mapped directly onto the coarse grid, or *full weighting*

$$(Rv)_i = \frac{1}{4}(v_{2i-1} + v_{2i} + v_{2i+1}), \quad i = 0, \dots \frac{N}{2} - 1,$$

where neighbouring points also have an effect on the coarse grid approximation.

On the coarse grid, we solve the system of linear equations

$$A^{(2h)}e^{(2h)} = r^{(2h)},$$

for example using the same smoothing method. We may choose the coarse grid operator $A^{(2h)}$ using the same discretisation scheme as on the fine grid. Alternatively, we can define it as

$$A^{(2h)} = RA^{(h)}I, \tag{2.33}$$

which is known as the *Galerkin property*. The Galerkin property together with

$$R = cI^T, \quad \text{for some } c \in \mathbb{R}, \tag{2.34}$$

are known as the *variational properties*. Whilst not desirable for all applications, they are useful for analysis [30].

The solution $e^{(2h)}$ is then interpolated back onto the fine grid

$$e^{(h)} = Ie^{(2h)}. \tag{2.35}$$

A common interpolation method for I is linear interpolation

$$(Iv)_i = \begin{cases} v_{i/2}, & \text{if } i \text{ is even} \\ \frac{1}{2}(v_{(i+1)/2} + v_{(i-1)/2}), & \text{if } i \text{ is odd} \end{cases} \qquad i = 0, \dots N-1.$$

Full weighting in combination with linear interpolation is a popular choice since (2.34) is satisfied.

After interpolating back to the fine grid, the approximate solution is then adjusted by the error estimate

$$\tilde{u}^{(h)} \leftarrow \tilde{u}^{(h)} + e^{(h)}.$$

A number of steps of an iterative smoothing process would then be applied again, this time with the initial guess of the updated solution $\tilde{u}^{(h)}$. This is called post-smoothing and marks the end of one iteration of the V-Cycle method. We repeat the steps described until convergence.

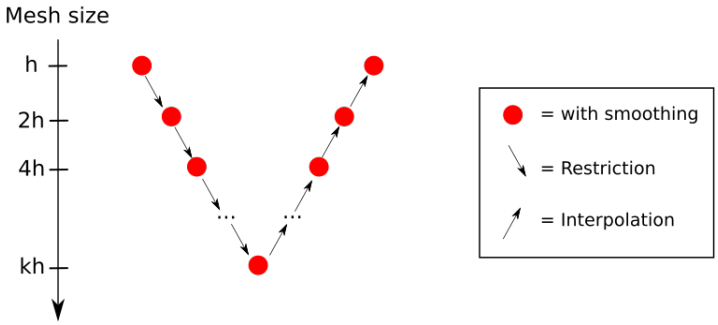


Figure 2.5: Generalised V-cycle

.

It is natural to extend this idea to more than two grids. The concept of a generalised V-Cycle is illustrated in Figure 2.5. However, we may want to utilise each grid multiple times per iteration. Another main type of multigrid methods is W-cycles, see Figure 2.6 for an example.

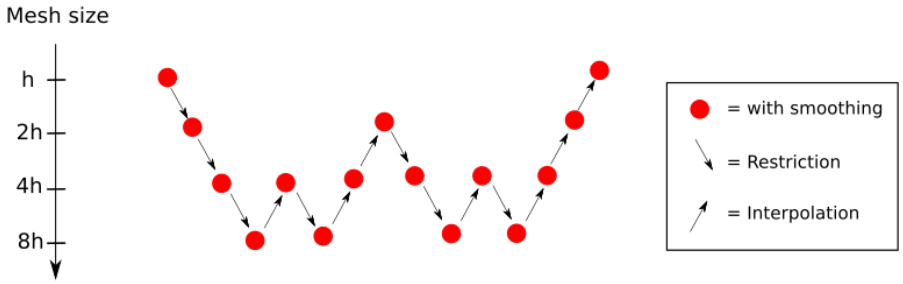


Figure 2.6: An example of a W-cycle.

As is the case with general iterative methods, a good starting guess will reduce the number of iterations required. But how do we start? The *Full Multigrid method* (FMG) starts on the coarsest grid (without any smoothing on the previous grids) then uses the interpolated solution as a starting guess for the second coarsest grid. From there, FMG essentially performs the V-cycle μ times for each grid which is not the finest grid, to give the starting guess for the next coarsest grid. See Figure 2.7 for an example with $\mu = 1$. The diagrams in this section give an idea as to the steps involved, the precise algorithms and pseudocode can be found for example in [28] or [30].

The processes discussed above were primarily developed to tackle spatially based problems and these are known as geometric multigrid methods. We conclude this section with a

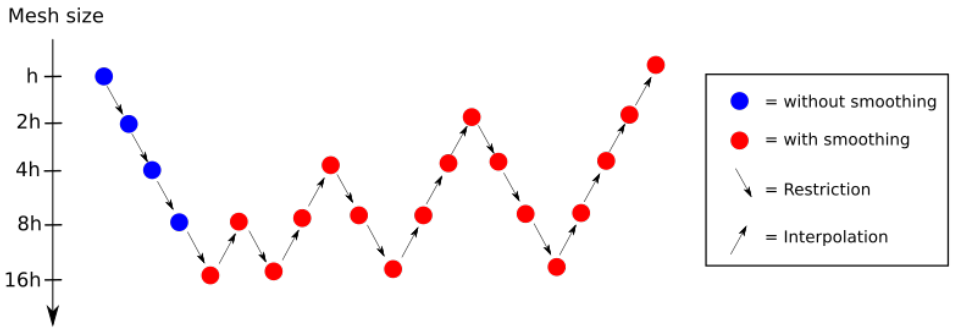


Figure 2.7: An example of the FMG method with $\mu = 1$.

quick word on algebraic multigrid method (AMG). AMG requires no inherent spatial grid, the problems can be entirely a sparse matrix problem. We can similarly define the key ingredients of algebraic multigrid methods. This is where definitions of the coarse grid operator such as the Galerkin condition (2.33) is really needed. Defining the restriction and interpolation operators for AMG is based on the non-zero entries of a matrix. Once the operators are set up, the conversion to the 'coarse' grid is automatic. No geometric information is used, this is particularly useful for problems with unstructured meshes. For more details see [67] and the references therein.

2.3.2 Multiscale methods in image analysis

Use of multiscale methods exist in machine learning. Since Alexnet [109], the use of convolutional neural networks (CNN) has been very popular for image analysis. The traditional convolution layers act like a restriction operator. Indeed various multiscale methods have been developed for image analysis.

Image analysis tackles a range of tasks, an immediate example is given by image classification. More complicated tasks build on classification. Object detection classifies items in the image and draws bounding boxes around the individual objects. Image segmentation splits the whole image into objects. Unlike object detection it does not use bounding boxes, the exact boundaries of each object are important. It can be thought of as a classification on each pixel of the picture. The parallel here with our work is that the output for segmentation is typically the same dimension as the input. Neural networks designed for such purposes use pooling techniques (which act as restriction operators) and upsampling methods (which act as interpolation operators).

There are multistream architectures for multi-scale image analysis, see [98]. The U-Net [157] uses a Fully Convolutional network [117], the high resolution input image is processed by convolutional layers and pooling layers down to lower resolutions, and then scaled back up. Each time an upsampling layer is performed, the output is concatenated with cropped versions of other resolutions. The authors demonstrate this architecture can learn from smaller sample sizes. Although other works may not explicitly state that they are multiscale, their principle is similar. These include the well known GoogLeNet [176] which uses inception modules. Each module utilises parallel 1×1, 3×3, 5×5 convolutional layers as well as a MaxPool layer (stride 1). The outputs of these parallel layers are then concatenated. Note that the 1×1 convolution is as introduced in [115].

Transfer learning [79] also assumes that the pre-training of weights on a similar task

will learn some low level common representation that can then be transferred and speed up training for new applications. We want to achieve a similar effect by using training on coarser grids to speed up training on fine grids.

2.3.3 Recent multigrid methods in machine learning

We proceed to review some other multigrid specific literature. Multigrid methods have evolved rapidly in the last few decades. In particular, we can solve nonlinear problems with Full Approximation Scheme (FAS) [30]. There have been applications of multigrid to the HJB (Appendix B) and Hamilton–Jacobi–Bellman–Isaacs (HJBI) equations which occur in differential games. For example FAS is applied to HJB and HJBI equations in [85].

With advances in machine learning, it is natural that there have been intersecting work between the two fields. We mentioned in Section 2.3.1 some common choices for interpolation and restriction matrices. The choice of these operators can change the rate of convergence, therefore efforts have been made on how to choose these operations. For example, in [101], the restriction matrix and the interpolation matrix are learnt with a neural network.

Convolutional layers can be thought of as combining local spatial information. Popular convolutional neural network (CNN) structures use multiple small filters and the dimension of the input is reduced through pooling layers. We have seen applications of this in Section 2.3.2.

Convolutions taken at earlier layers uses more local information on a finer grid, whereas convolutions taken at the later layers represent information on a coarser scale. Thus we can think of convolutional neural network as a multiscale method. This description seems to be the reverse logic to the FMG since popular CNNs such as AlexNet [109] and VGGNet [166] look on the fine scale first. On an intuitive level, viewing images from a coarse scale appears to be more akin to humans: we get the overall picture first before honing in on finer details. This motivated multigrid neural architectures as described in [102]. Each layer of a convolutional neural network has been decomposed into a pyramid. This allows faster information transfer spatially. In [88], the authors show how simple multigrid methods can be thought of as a convolutional neural network.

It has been noted that there are structure equivalents in particular between CNN and multigrid methods. As mentioned, pooling layers can be thought of as restriction operators. Techniques also exist for upsampling, they act as the interpolation operator. In moving towards understanding neural networks, Deconvolutional neural networks (deconvnet) [199] have been used to represent features in intermediate layers [198]. The deconvnet can be thought of as the interpolation operator. What remains is the coarse grid operator. For physical systems, as with geometric multigrid methods, the analogous idea is to define the equivalent problem on the coarse scale.

Chapter 3

Arrival management as a deep reinforcement learning problem

Now that the theoretical prerequisites have been established, we look to create a toy model that simulates a basic arrival management problem at Heathrow airport.

Due to the success of reinforcement learning in many domains, we cast the arrival management problem as a dynamic reinforcement learning problem. There has been existing work on reinforcement learning within air traffic control, examples include [31] and [142]. However these tackle a slightly different problem, namely conflict resolution. There the actions available to the agent are based on directions planes can move, rather than speed. We choose a discrete model to ensure the control space does not vary in dimensionality.

3.1 Our approach

We design a discrete toy model of the sequencing problem and apply reinforcement learning methods including classical methods such as Q-learning [188] and recent methods such as DDPG [114]. We model the air traffic controller as an agent that observes the number of aircraft and where they are, then sends out speed instructions. If an aircraft lands safely without violating minimum separation constraints then our agent gains a positive reward. Through repeated simulations, the agent can learn how to choose actions to increase the sum of expected rewards.

As we take finer discretisations of the problem, the dimensionality of the action space increases. Very quickly cataloguing all possible scenarios becomes computationally prohibitive if not impossible. In order to resolve this issue, we find a reduced representation of our model, and use neural networks as function approximators.

This approach gives us the flexibility to redefine the problem and impose restrictions from the physical problem, such as a minimum time between sending each aircraft a control, or only allowing actions to be sent at specified physical locations. With 100 repeats, we see the method often converges to the same local optimum when we place these restrictions on the control space.

On toy problems, the learnt model can successfully separate consecutive aircraft without further inputs. However, there are still cases when we do not converge; this motivated us to look for alternatives which are described later in the thesis.



Figure 3.1: Example view of our discretised system with the final state/cell on the right hand side representing the airport.

3.2 The model

Recall that one of the reasons we use concepts from reinforcement learning is that there is not a manual for the correct behaviour in air traffic control, only desired consequences – i.e. landing safely – much like a reward function. We think of the air traffic controller as our agent, who has the ability to choose how quickly to advance each plane. The environment describes the arrival and movement of aircraft.

We simplify the system of planes arriving at the airport by considering only the *radial distance* from the airport (ignoring the merging of four separate queues for now), this corresponds to assuming that the constraint at the terminal point is the only binding constraint. We discretise the radial distance into d intervals and neglect the height dimension. We assume that:

1. Aircraft take only two sizes.
2. At most one aircraft can occupy each interval (this is encouraged via the reward function and enforced through adjustments as detailed later).
3. Aircraft of different sizes should have a longer separation gap between landings compared to that of the same size – we do not impose asymmetry in the separation gap (a small aircraft followed by a large one is the same as a large aircraft followed by a small).
4. All aircraft have to move a minimum distance per timestep, and they are allowed to overtake each other.

Note here we are measuring the separation distance as the gap in time between consecutive landing aircraft.

To ensure that our problem is a Markov decision process, the state is given in the form: $S_t = [C_1, ..., C_d, L_s, L_t]$, where

$$C_i = \begin{cases} 0 & \text{if there are no planes in interval } i, \\ 1 & \text{if a small plane is in interval } i, \\ 2 & \text{if a large plane is in interval } i, \end{cases}$$

d is the total number of intervals, L_s is the size of the last aircraft that landed, and L_t is the time since the last aircraft landed (up to some defined maximum as we only need to know if there is a wake vortex present). We have set this maximum as half of the number of intervals, since policies may be different for busy and quiet periods. Figure 3.1 depicts our discretised system, the last column (on the right-hand side) represents the airport state. In this example $C = [C_1, C_2, ..., C_d] = [1, 0, 0, 1, 2, 0, 2, 1, 2]$.

We advance time in discrete steps and impose a minimum distance ($a_{\min}$) that each aircraft has to move per timestep. We use an episodic (finite horizon) setting as each day

forms a natural episode for real data. The length of an episode of simulation is based on the scale of discretisation we take (typically a multiplication factor of 10 compared with the resolution of time discretisation). The actions we can take at time t are $A^t = [A_1^t, A_2^t, ..., A_{d-2}^t]$, where each $A_i^t \in \{a_{\min}, ..., a_{\max}\}$ corresponds to the number of steps to advance the aircraft at interval i. For simplicity we take $a_{\min} = 1$, this also allows us to see if the choice of making more precise movements would increase rewards, as one might expect. In most cases $a_{\max}$ is set to $\lfloor d/10 \rfloor$, i.e. a tenth of the total distance from the airport (rounded down). Note that the action space has dimension $d-2$ since the last state represents the airport, and planes in the state prior to the last state will move deterministically into the airport at the next timestep. From this description it is clear an optimal policy will not be unique, since the action assigned to intervals with no aircraft is arbitrary, unless we introduce some kind of cost/regulariser.

Due to the constraint that at most one aircraft can occupy each interval, we model the arrival of an aircraft as a Bernoulli process with probability p. We choose p such that on average it should be possible to maintain the minimum separation gap under the given discretisation. The distribution of small and large aircraft at Heathrow is about 50/50. Therefore in our model, when an aircraft does arrive, it has equal probability of being either size.

We select a reward function that encourages the agent to choose actions that ensure the correct separation gap. Positive reward is given if the minimum separation gap is maintained as aircraft land. We can also impose some negative rewards to penalise undesirable behaviour. There is a negative reward if the agent chooses to perform an action that violates our modelling assumptions (e.g. more than one plane would end up at one interval). As these constitute illegal moves, in addition to the penalty, their effect is then adjusted automatically as follows:

1. We assume that our starting position is legal.

2. We can then always find another legal configuration, the most obvious one would be to advance all planes by one step.

3. To find the 'closest' action to that suggested by the agent, we update the states of the aircraft starting from the aircraft closest to the airport and working outwards.

4. If there is a clash (two aircraft in the same position), then we reduce the number of moves suggested for the more distant aircraft until there is no clash.

In this way, all possible conflicts would be resolved in one sweep. There is an option to have a negative reward for each time a plane does not land to ensure aircraft land as soon as possible. This is good for reducing delays but may not be the best for fuel efficiency; something a more advanced model would have to consider.

We apply the DDPG algorithm on this model. Since we have a discrete model and the DDPG is originally designed to be used on continuous action spaces, we make a slight modification. We mentioned that in [64], an approximate k-nearest neighbours look-up is used. Since this look-up is expensive, we assume that given a sufficiently fine discretisation, the continuous action, a, suggested by the agent is well-approximated by $\lceil a \rceil$, the ceiling function.

3.3 Results

Like all machine learning applications, our results are sensitive to the hyperparameters used; these are listed in Appendix D.

3.3.1 Comparison with table representations

An immediate question with using a nonlinear function approximator, like a neural network, is whether we can obtain a sensible policy without convergence guarantees.

Under certain stochastic approximation assumptions (2.16), SARSA and Q-learning are shown to be convergent. Therefore, as a first step we compare the performance of the DDPG against these classic table lookup methods.

The number of states/state-action pairs increases rapidly with the discretisation, therefore we are restricted to testing a small problem size. We split the space into 5 discrete intervals. The minimum separation gap between two aircraft of the same size is 0 and of different size is 1. This gives us a total of 972 states. Let $a_{\max} = 3$, this gives us 27 possible actions. We see that even on such a small scale problem we have 26,244 state-action pairs – which is why function approximation would be critical as we scale up. We use episodes of 200 timesteps.

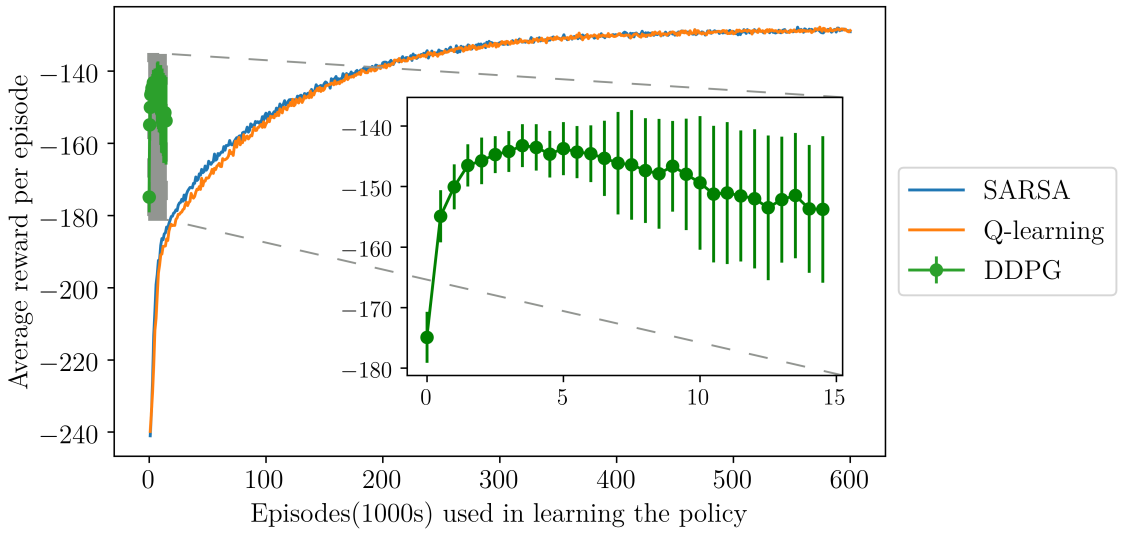


Figure 3.2: Average reward per episode as the policy is learnt using exact table representations with SARSA and Q-learning, and using neural networks in DDPG when we penalise lack of landings.

We can use a penalty for each timestep that an aircraft does not land to encourage faster landings. Note that the aircraft arrival model is still fixed so the system is not receiving any more aircraft, therefore we would expect that the average reward would decrease by the difference between the number of aircraft arrivals and the number of timesteps. Figure 3.2 shows that in this case the average reward per episode converges to around -130 after training on 600,000 episodes with SARSA and Q-learning. Also shown are the mean results of the DDPG implementation over 100 repeats, along with the standard deviations. We see that by using function approximation, we get much faster improvement but performance is varied, with some repeats performing better than others.

One comment to make here is that the learning rate is 0.95 and this is chosen after optimising over $[0.85, 0.90, 0.95]$. Since the rate is an endpoint of the range we optimised over, further improvements may be possible.

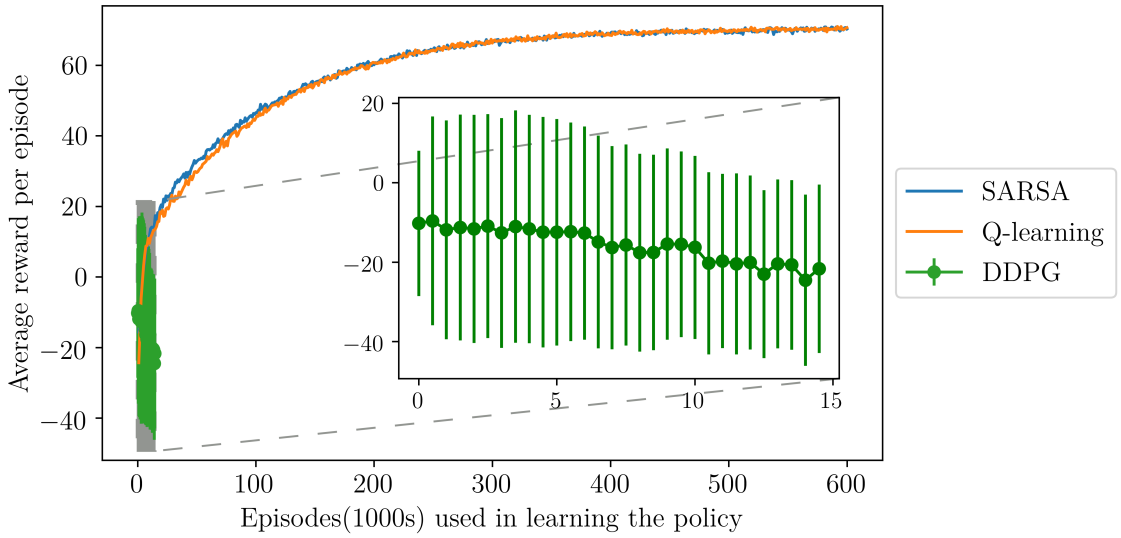


Figure 3.3: Average reward per episode as the policy is learnt using exact table representations with SARSA and Q-learning, and using neural networks in DDPG when penalisation of no-landing is not used.

If we do not penalise for lack of landings, then we get the results shown in Figure 3.3. We see that the average reward converges to 70 for the table representation methods. In contrast to the above case with the penalty, DDPG seems to struggle more with this reward function and the average reward per episode is less than 0, suggesting that the policy is choosing illegal moves or allowing aircraft to violate minimum separation gap. This implies that the reward signal may not be as strong in the case where we do not have the extra penalty.

We note that the performance of the DDPG can be quite variable. In Figure 3.4 and Figure 3.5, we see selected individual simulations that give good performance. The problem is therefore that this method is not stable, it does not consistently perform well for different initialisations. This is something that can be explored further. Given the speed of improvement for the case with penalty, the results from both cases would still be interesting to see going forward.

3.3.2 Restricted frequency of controls

In the toy example, we are free to issue an instruction at every time-point. This is impossible in practice, what happens if the instruction issued remains fixed for a period of time? We enforce this by expanding our state space to include the previous speed instruction issued $X = X_1, ..., X_d$, so that the Markov state space is now

$$S_t = [C_1, ..., C_d, X_1, ..., X_d, L_s, L_t].$$

The update of the state space is described in Algorithm 4. We see that we first update X if a new instruction A is given. Then we proceed with updating each plane as closely to

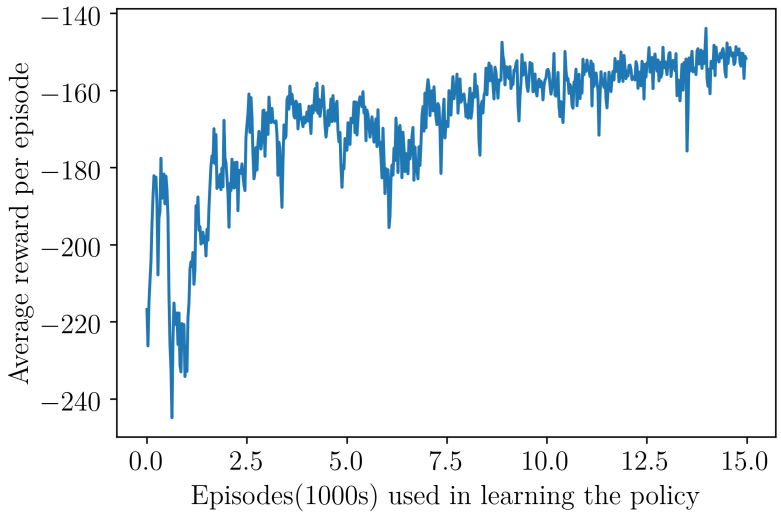


Figure 3.4: A single repeat of the DDPG method when we penalise lack of landings.

the location suggested by X as possible, incurring a penalty proportional to the amount of adjustment required. When the new location is determined, both the aircraft C_i and the instruction X_i is 'moved' to the new location. We can think of this as attaching a label to the aircraft that we see in Figure 3.1, this label determines how the aircraft moves and is attached to the aircraft until a new label is assigned after 5 timesteps.

Since we have less flexibility on the controls we can exert, we expect that this will lead to lower expected reward from the final policy. To test our hypothesis, we used a discretisation scale of 100 intervals and each instruction is repeated for 5 timesteps. We run 100 repeats to determine the average performance and standard deviation. See Appendix D for the full details. Note that with such a fine discretisation scale, it is no longer feasible to run SARSA and Q-learning for comparison as there are 3^100 possible values for the combination of aircraft that can be in each space.

The results of this new model can be compared with the original system. In Figures 3.6 and 3.7 we see the average performance for having repeated actions and the original set-up respectively. Contrary to the hypothesis, there does not seem to be much difference. This lack of difference means the policy learnt on the full system is not utilising the extra control. One reason for this may be due to the lack of convergence to the globally optimal policy. By having a -1 reward each time the plane does not land and having 1000 timesteps, we know it has converged to a local optimum since we would expect the average reward to be higher than -1000. It is interesting to see that the standard deviation drops as the policy is being learnt. The lack of variation is likely due to the same/similar policies being learned. The convergence is quicker for the full system as opposed to with repeated speed instruction. This may be due to the larger variance of rewards gained for each instruction when each action is repeated for 5 timesteps.

Figure 3.8 is generated with the same experimental set-up as above except a penalty is no longer imposed for no landing. This can be compared with results from the original system (Figure 3.9). We make similar observations to the case with penalty in terms of rewards and convergence: the rewards of around -500 is much lower than optimal, as a negative reward in this case implies that illegal moves are being made (which the system automatically adjusts) or minimum separation gaps are violated.

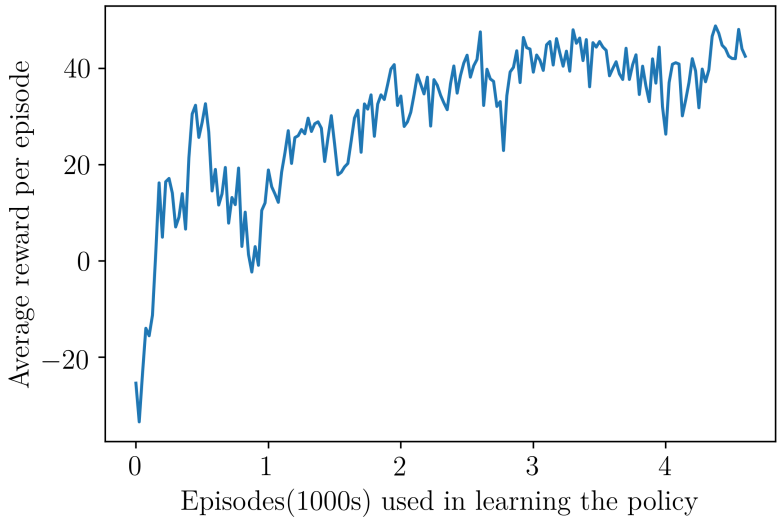


Figure 3.5: A single repeat of the DDPG method when penalisation of no-landing is not used

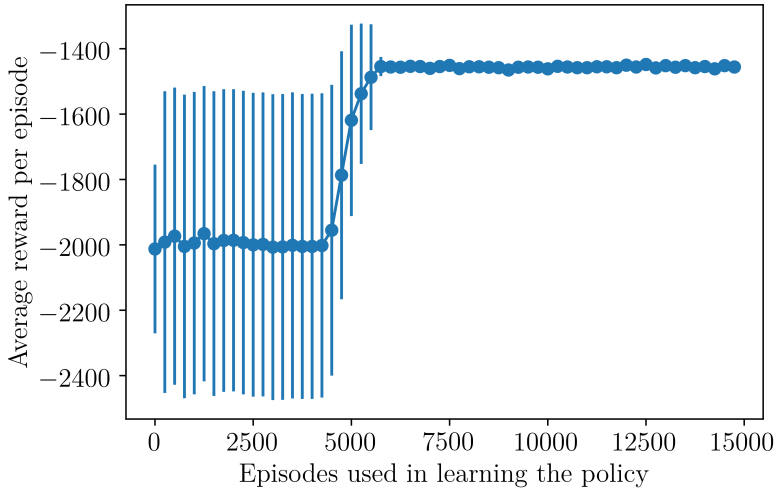


Figure 3.6: Average reward per episode as the policy is learnt with repeated instructions when penalisation of no-landing is used.

3.3.3 Restricted number of controls

We have assumed that controls can be made at every point. This micro-management of flights is also an unreasonable assumption. We will see in Part III a structured approach that NATS is looking at, under a system of metering points. In this chapter, we look at a practically feasible approach to issue an instruction as the aircraft enters the system, then again when the aircraft is at a fixed distance away from the airport and beginning descent. We set this point to 1/3 of the way from the airport/final interval, see Figure 3.10 for a visual representation. This is similar to what is done in practice at the moment. How would policies change on our model if we can exert controls at just two locations? To explore this question we still require the enlarged state space that includes the last speed instructions

Algorithm 4: State update under restricted frequency of control

Input: Current state $S = [C_1, ..., C_d, X_1, ..., X_d, L_s, L_t]$, an optional new speed instruction $A = [A_1, A_2, ..., A_{d-2}]$.

if *New instruction given* **then**
  for i *in* $1, ..., d-2$ **do**
    $X_i \leftarrow A_i$.
  end
end

Planes in penultimate slot move deterministically:

$C_d \leftarrow C_{d-1}, \quad X_d \leftarrow X_{d-1}$

if $C_d \neq 0$ **then**
  $L_t = 0, \ L_s = C_d$
else
  $L_t = L_t + 1$, up to a maximum.
end

for i *in* $d-2, \ldots, 1$ **do**
  if $C_i \neq 0$ **then**
    for j *in* $\min(i + X_i, d), \min(i + X_i, d) - 1, \ldots, i$ **do**
      if $C_j = 0$ **then**
        That is, if there are no flights in this slot, move aircraft
        $C_j \leftarrow C_i, \quad X_j \leftarrow X_i$.
        Add a penalty for illegal moves
        $R \leftarrow R + j - (i + X_i)$.
        break
      end
    end
  end
end

Simulate new arrival of aircraft to update C_1. Return new state S.

seen in Section 3.3.2. Having only two control points is a large decrease in the dimension of our control space and therefore we expect a much lower average reward from the final policy learnt.

We see the results of the two-control case, for a system with a discretisation of 100 states, with penalty in Figure 3.11. Interestingly, we do not see policies across different repeats converge on the same local minima – the standard deviation bars remain large. When we compare this with the original set-up in Figure 3.7, we see that the average reward is lower as we expected. This shows the algorithm does make some use of the additional actions allowed at each interval for the original set-up.

In Figure 3.12 we see the results for two control points when there is no penalty for timesteps without any flights landing. The average reward is again a lot lower in comparison to the case where we can make an action at every interval (Figure 3.9). We make the observation that the standard deviation is also not decreasing in this case. This means that the algorithm is not finding the same policy when we restrict the action space to just two controls. This may be due to having many contributing factors for the reward function, resulting in large variance. The reward signal of each action is harder to learn when you have to make the decision for each aircraft well in advance of arrival.

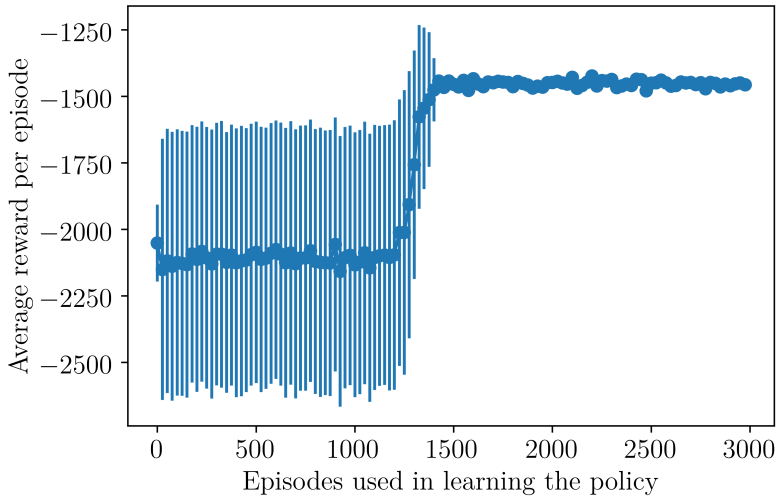


Figure 3.7: Average reward per episode as the policy is learnt on the original set-up when penalisation of no-landing is used.

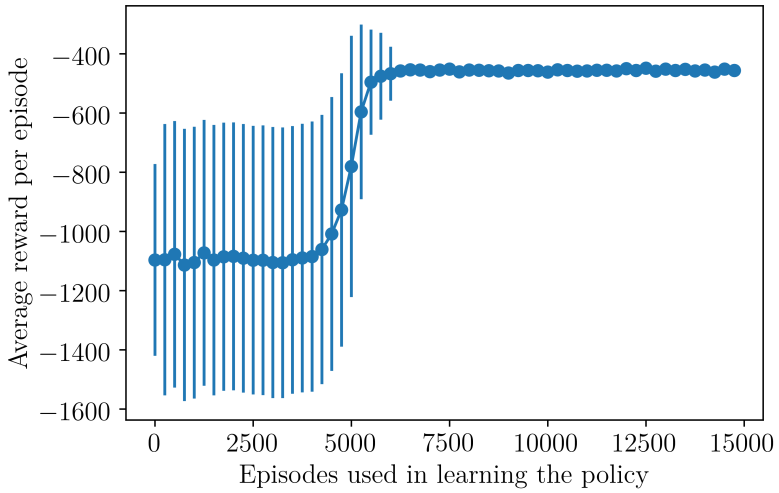


Figure 3.8: Average reward per episode as the policy is learnt with repeated instructions when penalisation of no-landing is not used.

However this behaviour may also be saying something about DDPG. DDPG is a method originally designed for continuous action spaces, but which are not typically high dimensional. We note that both the toy problem with an action space of dimension 3 and the two-control case still show variability across different repeats as we train. This means that there is a greater probability of discovering a useful policy in these cases, since we continue to explore and search for better policies in each repeat. It is when we have a large control space of 98 dimensions that we see all the policies converging rapidly to one local minima. Why does this happen? One possibility is that in 98 dimensions, the reward landscape can be quite flat, each action may not have a large effect, especially in a sparse system. Therefore once we converge to a local optimum, it is difficult to escape.

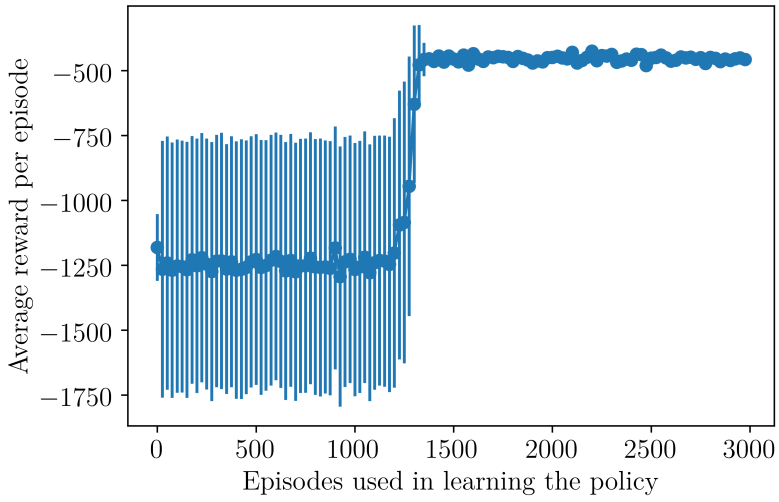


Figure 3.9: Average reward per episode as the policy is learnt on the original set-up when penalisation of no-landing is not used.

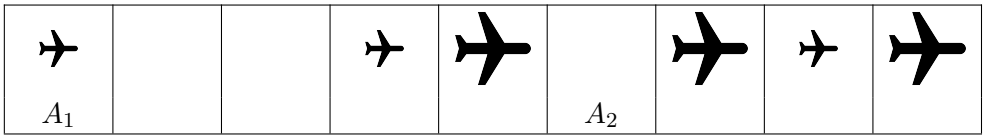


Figure 3.10: Example of two-control system, speed of aircraft changed at A_1 and A_2.

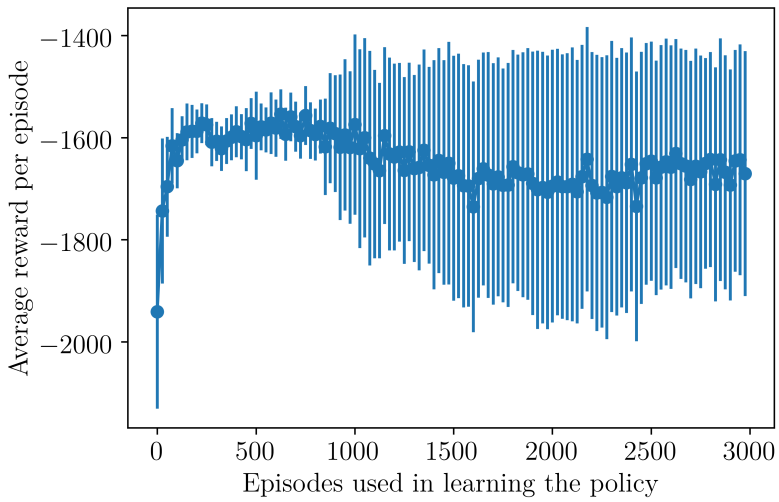


Figure 3.11: Average reward per episode of 100 repeats for two controls when we do penalise lack of landings.

3.3.4 Results with path signature features

In the previous section, we feed into our neural network the size and location of planes in the system. This is combined with the size and time of the last aircraft that landed. Another question we want to address is whether we can pass in some features instead to reduce the

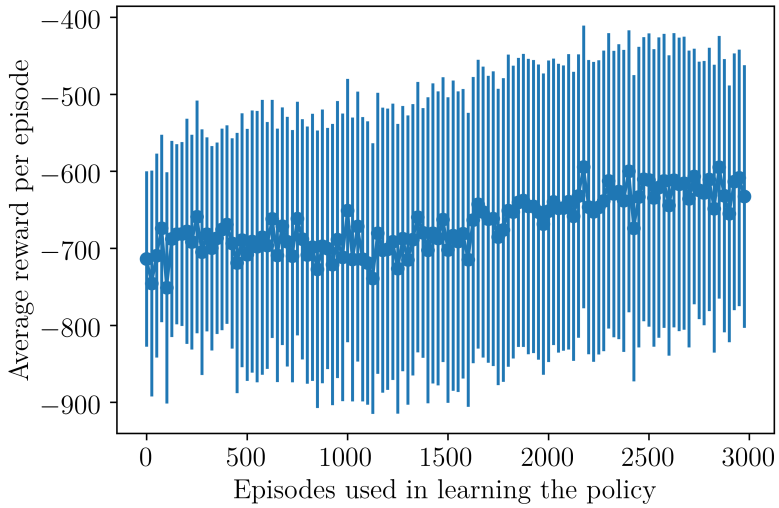


Figure 3.12: Average reward per episode of 100 repeats for two controls when we do not penalise lack of landings.

dimensions of the input.

Since we would ideally like to have long sequences of aircraft of the same size, intuitively the graph of the aircraft size plotted against distance from the airport should have long flat regions. Since the path signatures described in Section 2.2 captures geometric properties of paths, we extract some truncated signatures to use as features. We programme a processor to convert the raw size and location data into signatures. Using this processor, our replay memory no longer stores $(s_t, a_t, r_t, s_{t+1}, a_{t+1})$ but instead stores $(f(s_t), a_t, r_t, f(s_{t+1}), a_{t+1})$; that is the states are replaced with the features. It was mentioned in [154] that the log signature is a more compact representation of the signature and we compare performance of the DDPG algorithm of both truncated signatures and log signatures. The results from 100 repeats can be seen in Figures 3.13 and 3.14 for the cases with and without no-landing penalty respectively.

We see that training with the truncated signatures and log signatures converge to the same average reward per episode as the full state (Figures 3.7 and 3.9). This suggests that in all cases, similar policies are being learnt, and that whatever led to the chosen policy in the full state is also captured by these path signatures. The rate of improvement is faster for the signatures compared with the log signatures and the original set-up. Although we have reduced the input dimensionality, the control space is still the same size. Therefore our discussion regarding flat surfaces for large sparse systems is still valid in this case.

3.3.5 Multigrid-inspired RL

We want to exploit the structure of neural networks to improve/speed up the training process for reinforcement learning problems so that large action-space dimensions can be used. Can we make use of parallels in techniques between multigrid methods and machine learning as discussed in Section 2.3 to improve performance and learn policies faster?

Since the problems we are addressing are physical with an inherent spatial grid, as with geometric multigrid methods, we can define equivalent problems on coarser scales. Inspired by other work in the literature [102, 88], our idea is based upon FMG (Section 2.3.1). With

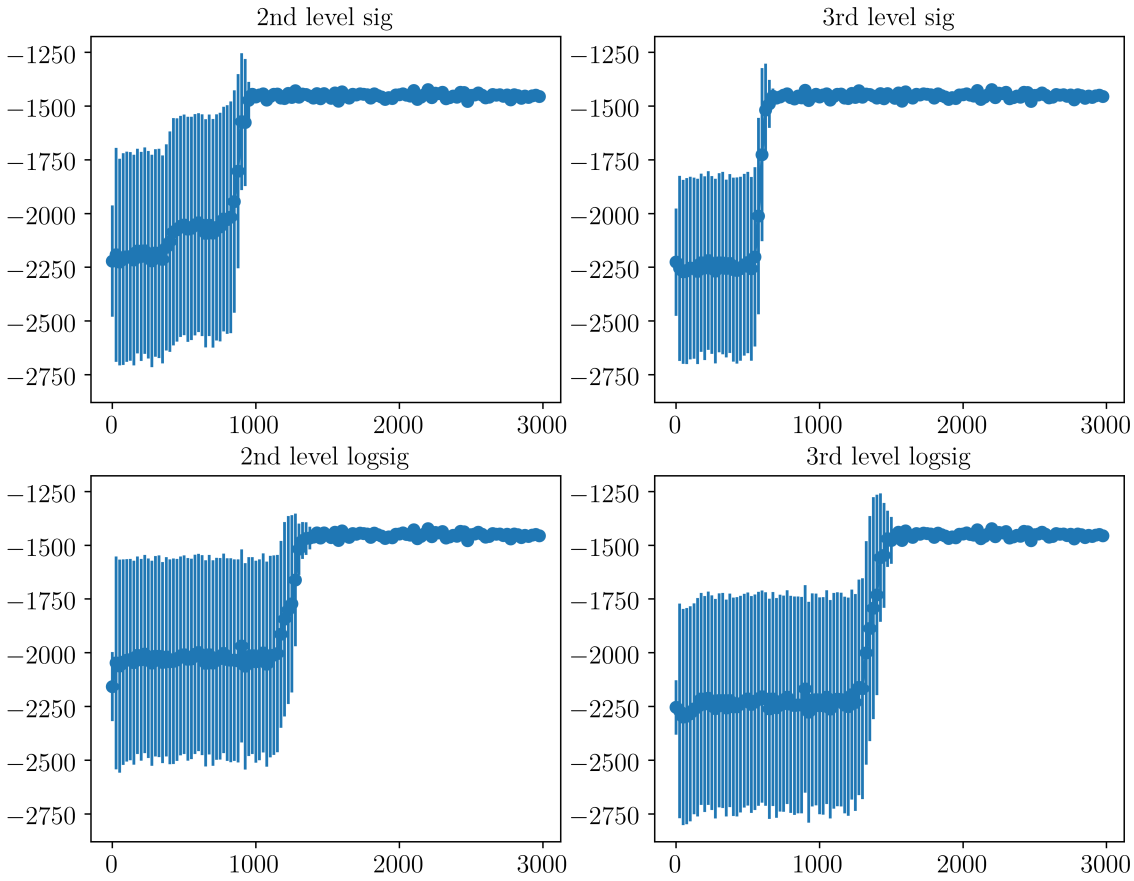


Figure 3.13: Average reward per episode as the policy is learnt using truncated signatures and log signatures when we penalise lack of landings.

the neural network, we can first restrict the problem onto the coarsest grid by a pooling operation (e.g. a MaxPool layer). The trained policy/parameters can be fed into another layer of the neural network which operates at a higher granularity after an upsampling layer, and we can continue in a similar fashion until the desired scale. See Figure 3.15 for an illustration of this idea.

As we wish to improve training policies in deep RL, we utilise this idea for the actor network which is consuming the information from the environment to learn policies. Here it is natural to perceive there are valuable information at multiple scales (as with the image processing applications discussed in Section 2.3.2).

In Figure 3.16 we see the results of using a multi-grid structure on the penalised problem. Compared with Figures 3.7 and 3.13, we do see a faster convergence, though the final value of the policy does not seem to improve, so the same/similar local optimum is reached.

The precise architecture of the neural network used for these can be seen in Figure E.1 and Figure E.2 for the critic and actor networks respectively. Note that the multigrid-inspired structure is used for the actor network as the focus here is on policy improvement, but it can also be interesting to explore such structures for the critic network too (for policy evaluation).

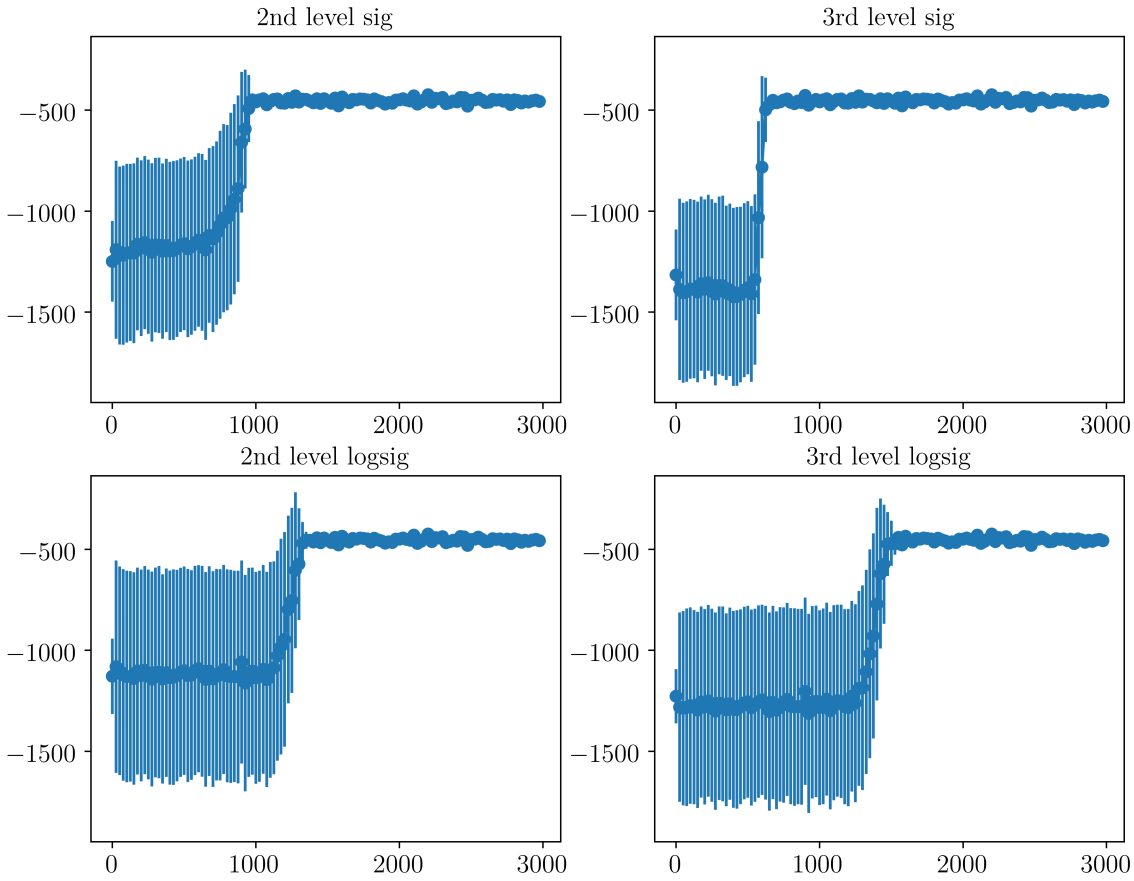


Figure 3.14: Average reward per episode as the policy is learnt using truncated signatures and log signatures when penalisation of no-landing is not used.

3.4 Stability of deep RL methods

We have seen in the two-control case, there is a lot of variability across the different runs. In this section, we discuss the stability and convergence analysis of RL methods. We begin by looking at the existing results from the literature. Then we examine the problem of stability in detail by looking at a simple linear quadratic example, and provide some qualitative understanding for the instability behaviour. Finally, we discuss regularisation and its wide use amongst popular deep RL methods.

3.4.1 Summary of the key methods and theoretical results in the RL community

As discussed in Section 2.1, if the state transition function and the reward function are known/estimated, then these belong to the category of 'model-based' methods. The classic value-iteration and policy-iteration generally fall into this class, which is also known as *planning* methods. Model-free methods that learn from sample transitions of the Markov decision process (MDP) without explicit estimation of the state transition function are popular with the RL community. Model-free algorithms tend to be computationally cheaper to compute and have better asymptotic performance [145], but model-based methods are

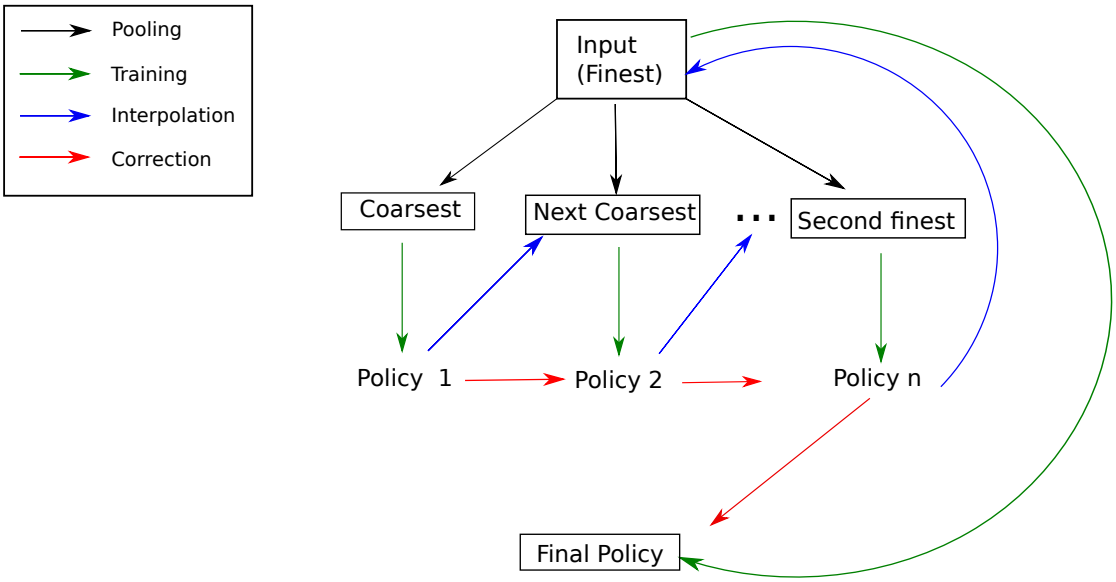


Figure 3.15: Visualisation of 'multigrid' neural network architecture.

more sample efficient [153].

Earlier literature primarily focused on finite MDPs, which can be represented as look-up tables. Model-free, value function based methods such as Q-learning and SARSA were hugely popular. Proofs for the convergence of table/exact Q-Learning [94, 179] show that we have a contraction (in $||\cdot||_\infty$ or $||\cdot||_v$ for some vector v). In particular, the Bellman optimality operator, given by

$$TQ(s,a) = f(s,a) + \gamma E_{s'}\Big[\max_{a'} Q(s',a')\Big], \tag{3.1}$$

where f is the reward function, is a γ-contraction in the infinity norm (for $0 < \gamma < 1$). A standard assumption in these convergence proofs, which follows from the Robbins–Munro stochastic approximation theory [156], is that the step sizes satisfy

$$\sum_{n=1}^{\infty} \alpha_n = \infty, \quad \sum_{n=1}^{\infty} \alpha_n^2 < \infty \tag{3.2}$$

(note that this is a reproduction of (2.16) for convenience). For the 'table-lookup' methods of Q-learning and SARSA, as long as all states are updated infinitely often and step sizes satisfy (3.2), then we have convergence for policy evaluation.

In the case of a finite horizon and finite MDP, we can find a linear function approximation of the value function [178], $V(s,w) = w^T X(s)$, with Monte Carlo on-policy sampling by choosing parameters w that would minimise

$$E[(V^\pi(s) - V(s,w))^2].$$

Here $X(s)$ is a vector of features and the expectation is taken over the stationary distribution of the states. Recall that the Monte Carlo update method means that we use the observed sum of (discounted) rewards for the full 'episode' in the place of $V^\pi(s)$. In the case of temporal difference updates, $f(s,\pi(s)) + \gamma V(s',w)$ is used to estimate the return. There are theoretical results to show that the approximation is stable around the minimiser [174].

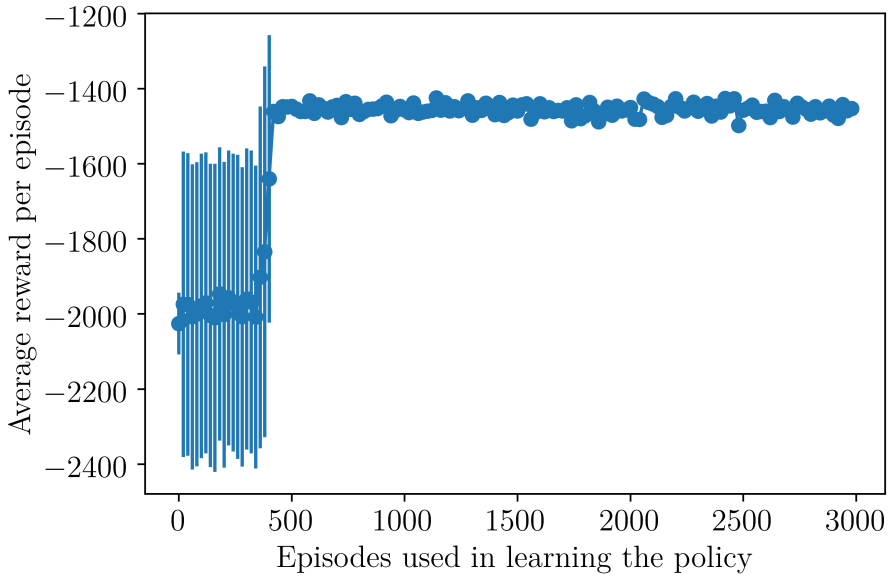


Figure 3.16: Average reward per episode for policies learnt with a multi-grid inspired neural network on the penalised problem.

Recall that instead of approximating the value function, we can also approximate the policy. The idea is to search over some probability distributions for the optimal policy by assuming the optimal policy is from a class of policies parameterised by e.g. θ.

Policy gradient methods became popular in the 90s with the 'Policy gradient theorem' (Theorem 2.1.5) where a rewriting of the gradient of the objective with respect to the policy parameters hides the explicit dependence on the state transition, allowing use of sample-based 'model-free' methods. Suppose that we have an objective function, $J(\pi_\theta)$, which is often the expected sum of discounted reward from the starting state, then

$$\frac{\partial J}{\partial \theta} = \sum_s d^{\pi_\theta}(s) \sum_a \frac{\partial \pi_\theta(s,a)}{\partial \theta} Q(s,a), \tag{3.3}$$

where d is the stationary distribution for the given policy.

A function approximator is said to be *compatible* if

$$\frac{\partial Q^w(s,a)}{\partial w} = \frac{\partial \pi_\theta(s,a)}{\partial \theta} \frac{1}{\pi_\theta(s,a)}. \tag{3.4}$$

In [175], the authors show that for compatible function approximators, we have

$$\frac{\partial J}{\partial \theta} = \sum_s d^{\pi_\theta}(s) \sum_a \frac{\partial \pi_\theta(s,a)}{\partial \theta} Q^w(s,a).$$

This says that the policy gradient theorem is still exact when we use a compatible function approximator for $Q(s,a)$. Note that the condition (3.4) is obviously satisfied in the case of linear function approximators given by

$$Q^w(s,a) = w^\intercal \partial_\theta \log \pi(s,a), \tag{3.5}$$

and indeed this form is used by many subsequent papers to prove various results including for the natural policy gradient [96]. The authors of [175] noted that this compatibility condition (3.4) might only be satisfied in the case of linear function approximators.

Natural gradients [13] are the proposed search directions under the assumption that the underlying parameter space is not a Euclidean space but some general curved Riemannnian space with some Riemannian metric tensor G. The steepest descent direction of a general loss function $L(w)$ is then given by

$$-\tilde{\nabla}L(w) = -G^{-1}(w)\nabla L(w).$$

The Riemannian structure of the parameter space of a statistical model or a multilayer neural network is defined by the Fisher Information.

The idea of natural gradients were utilised by Kakade [96] in what is known as a *natural policy gradient*. This concept defines a metric based on the policy parameter space. In [96], finite MDPs (i.e. finite state and action space) were considered, with all policies considered to be ergodic and induces a distribution on the states. We seek to find $d\theta$, with $|d\theta| = \epsilon$ for some chosen ϵ, such that the objective function $J(\pi_{\theta+d\theta})$ is minimised. Suppose $|d\theta|^2 = d\theta^\intercal G(\theta) d\theta$, then

$$-G^{-1}(\theta)\nabla J(\pi_\theta) \tag{3.6}$$

is the steepest descent direction. By using the Fisher Information, we will have a positive definite matrix for G, and so in [96]

$$G(\theta) = E_{s\sim d^{\pi_\theta},\, a\sim\pi_\theta(s,a)}\left[\frac{\partial \pi_\theta(s,a)}{\partial \theta_i}\frac{\partial \pi_\theta(s,a)}{\partial \theta_j}\right].$$

When the natural policy gradient is used in conjunction with a compatible function approximator for $Q(s,a)$ satisfying (3.4), i.e. a linear $Q^w(s,a)$ as seen in (3.5), choosing w to be the steepest descent direction of the policy (3.6) minimises the expected square approximation error between the action-value function $Q(s,a)$ and its approximation. Furthermore, for an affine parameter space for θ, a sufficiently large step in the natural gradient direction is equivalent to a greedy policy improvement step. For non-affine spaces, the update in θ is towards direction of policy improvement.

Regularised algorithms based on natural policy gradients are very popular, these include Trust Region Policy Optimisation (TRPO) [162] and Proximal Policy Gradient (PPO) [163]. TRPO regularises the problem with the Kullback–Leibler (KL) divergence, so the optimisation problem is formulated as

$$\begin{aligned}&\max_{\theta_{k+1}} L_{\theta_k}(\theta_{k+1})\\ &\text{subject to } \bar{D}_{KL}(\theta_{k+1},\theta_k) < \delta,\end{aligned}$$

where

$$\begin{aligned}L_{\theta_k}(\theta_{k+1}) &= J(\pi_\theta) + \sum_s d^{\pi_k}\sum_a \pi_{k+1}(s,a)(Q_{\pi_k}(s,a) - V_{\pi_k}(s)),\\ \bar{D}_{KL}(\theta_{k+1},\theta_k) &= E_{s\sim d^\pi}[D_{KL}(\pi_{\theta_{k+1}}||\pi_{\theta_k})].\end{aligned}$$

This form of L can be thought of as an approximate improvement of θ_{k+1} over θ_k. It is shown in [162] that minimising some surrogate objective function guarantees policy improvement, which motivated the TRPO algorithm. This problem can be approximately solved using

the conjugate gradient algorithm, by making a linear approximation to the objective and a quadratic approximation to the constraint. The search direction is approximated by finding the quadratic approximation to the KL divergence, $\bar{D}_{KL}(\theta_{k+1}, \theta_k) \approx (\theta_{k+1} - \theta_k)^\intercal A(\theta_{k+1} - \theta_k)$, where A is the Fisher Information matrix.

One limitation of TRPO is that computing A is expensive. As an alternative, PPO [163] uses soft constraints. That is, the problem is rewritten as a penalty method so that first order methods can be used. Further constraints are imposed for stability by clipping or penalising the ratio $\pi_{k+1}(s,a)/\pi_k(s,a)$ to limit large changes from the current policy.

In the last few years, there has been some renewed interest in the convergence/rate of convergence of policy gradient methods and a flurry of activity ensued. Of particular interest is [11], they summarised the state of the play as: "*folklore guarantee is that these methods converge to a stationary point of the objective, assuming adequate smoothness properties hold and assuming either exact or unbiased estimates of a gradient can be obtained (with appropriate regularity conditions on the variance). However, this local convergence viewpoint does not address some of the most basic theoretical convergence questions, including: 1) if and how fast they converge to a globally optimal solution (say with a sufficiently rich policy class); 2) how they cope with approximation error due to using a restricted class of parametric policies; or 3) their finite sample behaviour.*"

By assuming policy evaluation with exact policy gradients, Agarwal et al. [11] were able to obtain convergence rates for projected gradient ascent. Another state distribution is introduced as a replacement for exploration. The proof of the convergence of the softmax parametrisation depends on all states being visited with positive probability which only works for finite spaces. Without regularisation, it is possible to show convergence by relying on the value function being increasing/monotone pointwise, that is

$$V_{k+1}(s) \geq V_k(s).$$

Another result of interest in [11] is that they obtain sublinear convergence rates for function approximation with log-linear policies (softmax of linear functions) or satisfies the β-smoothness condition

$$||\nabla f(x) - \nabla f(y)||_2 \leq \beta ||x - y||_2.$$

Bhandari and Russo [25] showed with larger steps it is possible to obtain linear rate of convergence for the tabular case. However in both [11] and [25], finite state and exact policy evaluation is asssumed. One paper which examines the case of approximation through samples is [37]. They showed that a softmax parameterisation of the action-value function converges linearly and the result is stable under approximation.

A unified way to look at RL methods, such as TRPO and PPO, is that they are simply solving regularised MDPs. This case of a general regularised problem was presented in [75]. Whilst their proofs are for the finite state case, they claim that their results can 'extend to more general cases' (though they do assume that a greedy step can be performed for the policy update).

The general case of an MDP regularised by some strongly convex function retains all the 'nice' properties of the Bellman operator (e.g. contraction in infinity norm) [75]. The regularised value function is then the unique solution to the regularised Bellman operator, and this regularised value function can be shown to be 'close' to the unregularised version provided the upper and lower bound of the regularisation is small. There is also a worse case bound for how sub-optimal the policy of the regularised case is.

For the modified policy iteration (originally proposed by [149]), [75] assumes that we can use some greedy step to find an updated policy which is less than some bound away from the optimal (regularised) policy. This loss can then be tracked and shown to be reducing as the number of steps increases.

For actor-critic methods with function approximation, in [24] the authors show that a version of policy iteration with a general differentiable function approximator is convergent to a locally optimal policy. They require the policy class to be closed under policy improvement (linear to linear etc). Convexity of $\mathcal{A}(s)$, and $Q(s,a)$ w.r.t. a are assumed along with other regularity conditions. These assumptions ensure that there are no spurious local optima by constructing the update of the gradient descent step towards the policy gradient. Since there are no spurious local minima, the convergence would be to a global optimum.

In [165], the deterministic policy gradient (DPG) was derived

$$\nabla_\theta J(\pi_\theta) = E_{s\sim d^{\pi_\theta}}[\nabla_\theta \pi_\theta(s) \nabla_u Q(s,a)|_{a=\pi_\theta(s)}]. \tag{3.7}$$

This was a significant result as the authors show this can be estimated more efficiently than the stochastic policy gradient.

In the off-policy case, a term relating to $\nabla_\theta Q(s,a)$ was dropped as this is difficult to estimate. This was justified by citing Degris et al [60], which says that for sufficiently small step sizes, we can still guarantee policy improvement. Furthermore, if the approximating class contains the true value function, then the set of stationary points of the approximate gradient will contain the set of stationary points of the true gradient. A later errata [61] seems to suggest their work only applies to the tabular case and not to general function approximations.

The DPG (3.7) was then used (in an off-policy way) in the DDPG algorithm [114] though theoretical guarantees are still limited for non-linear function approximators like a neural network. For recent developments see [186].

We provide an analysis of a simple LQ set-up in Appendix F to provide some intuition and example of why these RL methods may be failing. In particular, we look at how important monotonicity and having an upwind method is for the stability of numerical methods, and what can happen if the method does not possess such properties.

3.4.2 Regularisation

To recap, for state-action value based algorithms where the policy is recovered from the state-action value, one cause of instability is that small changes in the value function can result in big changes in policy. As a result, popular algorithms in RL involve regularisation techniques [162, 163] to help with stability. As future work, it may therefore be interesting to see how such regularised methods (PPO, TRPO) perform on our problem set-up.

Observe that regularisation can also be thought of as a way to convexify the problem. In the next chapter, we see an alternative modelling approach using entropic regularisation.

Chapter 4

Summary

To summarise, in Part I, we looked at how to model our system as a discrete deep reinforcement learning problem. We compared using DDPG to SARSA and Q-learning. Despite the quicker initial gains for DDPG, the final reward of SARSA and Q-learning are higher. However, these methods cannot be applied when we take a finer discretisation.

We looked at several variations of our discrete model and used a discretisation of 100 intervals to generate results. Considering some physical constraints, we tested cases where we repeated each action for multiple timesteps or apply actions at only two intervals. The rewards gained from the two-control case is much lower than the original set-up, although there is little difference for the repeated instructions case. Finally, we reduced the input dimension to the neural networks by extracting path signatures. The results in this case are comparable with the full system.

When we ran 100 repeats of our model, the standard deviation across repeats can drop dramatically, suggesting the same or similar policies are learnt. This occurs when the control space has 98 dimensions and the system is very sparse. Although greater performance variability may be the cause for seeing greater variability in policies across repeats for the two-control case, it may also be due to the low dimensionality of the control space. By not getting stuck at a local minima as observed on the original set-up, this has the potential to improve further. The different behaviour observed for low action dimensions motivates the idea of using a multi-level/multigrid scheme, where we find the solution for a coarse discretisation, then use this to improve the implementation on a finer grid. Using such a neural network structure, we see that policies converged quicker.

However, we demonstrated that the application of DDPG on a simple linear-quadratic problem can still show instability. As part of our analysis, we offer one possible explanation. The problem of stability and convergence of deep reinforcement learning remains an active area of research.

Part II

Optimising (air) transport with optimal transport theory

Chapter 5

Background

5.1 Motivation

The global picture of our arrival management problem is that Arrival Managers (AMAN, Section 1.2.1) receives information, with which it computes predictions as to when each aircraft is currently scheduled to arrive; that is, we have an initial distribution of arrival times for each aircraft. The task is to map this to an alternative, improved distribution satisfying separation constraints. Finding such a map between distributions resembles an optimal transport problem.

Instead of modelling the air traffic controller as an agent, we can model each individual aircraft as an agent, who is trying to optimise their own objective with the range of actions available to them — an n-player game. The overarching goal is to minimise the overall conflict in the system while introducing minimal delays. For this, techniques from continuous optimisation such as block-gradient descent will be useful.

In this chapter, we outline some prerequisites to this alternative approach for modelling the arrival management problem in air traffic control. In particular, we overview materials from optimal transport and optimisation.

5.2 Optimal Transport

5.2.1 Introduction

We begin by giving a brief overview of the (discrete) optimal transport problem [113, 141].

The probability simplex is given by

$$\mathcal{S}_n = \left\{ x \in \mathbb{R}^n \,\middle|\, \sum_{i=1}^{n} x_i = 1, x_i \geq 0, \forall i = 1, \ldots, n \right\}. \tag{5.1}$$

Let us denote general discrete measures as

$$\alpha(x) = \sum_{i=1}^{n} a_i \mathbb{1}_x(x_i), \tag{5.2}$$

where we impose the constraint that $a_i \geq 0$, $\forall i$. If $\sum_i a_i = 1$, then this measure lives on the probability simplex.

In the original problem considered by Monge [124], we wish to move from one distribution to match another in an 'optimal' way. We give the formal definition for discrete problems below.

Let us take two spaces $\mathcal{X}$ and $\mathcal{Y}$ on which there exist some notions of distance, and define two measures $\alpha(x) = \sum_i^n a_i \mathbb{1}_x(x_i)$, $x \in \mathcal{X}$ and $\beta(y) = \sum_i^n b_j \mathbb{1}_y(y_i)$, $y \in \mathcal{Y}$. Let $c(x, y) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^+$ be a cost function.

Let T be a mapping from $\mathcal{X}$ to $\mathcal{Y}$. If T is such that

$$b_j = \sum_{\{i:\, T(x_i)=y_j\}} a_i, \tag{5.3}$$

then we say that β is the *pushforward distribution* of α and denote it as

$$T_{\#}\alpha = \beta.$$

The pushforward operator, $T_{\#}$, preserves positivity and total mass, and is linear (as a map on measures). For the Monge formulation of optimal transport problems, we search over all mappings that satisfy (5.3) to minimise the cost function, that is

$$\min_T \left\{ \sum_i c(x_i, T(x_i)) : T_{\#}\alpha = \beta \right\}. \tag{5.4}$$

For the Monge problem each particle is mapped to exactly one location; 'particle splitting' is not allowed. This means that there are problems without any solutions. See Figure 5.1 for a simple example of this.

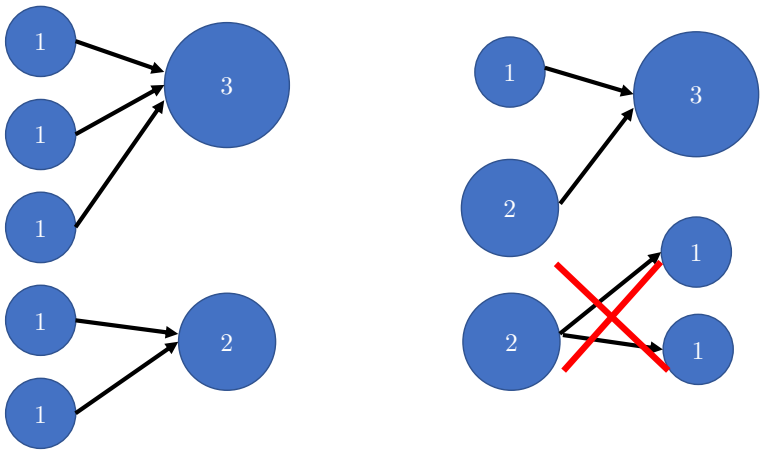


Figure 5.1: Two particle examples of optimal transport. On the left, we see a feasible mapping where 3 particles of mass 1 have been mapped to a particle of mass 3, and two particles of mass 1 have been mapped to a particle of mass 2. On the right, the set of feasible mappings is empty. Although we can map a particle of mass 1 and another of mass 2 to form the particle of mass 3, there is no way to obtain two particles of mass 1 under the Monge formulation ('particle splitting').

The feasible set above is non-convex and obtaining a Monge map is combinatorially difficult. This restriction on the solution space was relaxed in the works of Kantorovich [99], who approached the problem as a linear program on a convex polytope. Rather than a map,

we instead find a *coupling* between two measures. In the discrete case, the set of *coupling matrices* is given by

$$U(\boldsymbol{a}, \boldsymbol{b}) := \{P \in \mathbb{R}^{m \times n} : P\mathbb{1}_n = \boldsymbol{a}, P^\intercal \mathbb{1}_m = \boldsymbol{b}\}, \tag{5.5}$$

where $\mathbb{1}_n$ is the vector of ones of length n and $\boldsymbol{a} \in \mathbb{R}^m$, $\boldsymbol{b} \in \mathbb{R}^n$ have elements a_i, b_j as given in $\alpha(x)$, $\beta(y)$. The matrix P may also be referred to as the *transport plan*.

Since the set $U(\boldsymbol{a}, \boldsymbol{b})$ has $n + m$ equality constraints and is bounded, it is a convex polytope. Observe that if $P \in U(\boldsymbol{a}, \boldsymbol{b})$ then $P^\intercal \in U(\boldsymbol{b}, \boldsymbol{a})$. Let us denote the cost matrix by C and the Frobenius inner product as

$$\langle C, P \rangle := \sum_{i,j} C_{ij} P_{ij},$$

then the Kantorovich problem is the linear program given by

$$L_C(\boldsymbol{a}, \boldsymbol{b}) := \min_{P \in U(\boldsymbol{a}, \boldsymbol{b})} \langle C, P \rangle. \tag{5.6}$$

To make the measures explicit, we may also use $\mathcal{L}_C(\alpha, \beta)$ to denote this cost instead. Note that since this is a linear program, the solution is not necessarily unique.

Let $\mathcal{X} = \mathcal{Y}$ and consider the *ground cost* $c(x, y)$ as fixed. Suppose $c(x, y) = d(x, y)^p$, where $p \geq 1$ and d is a metric on $\mathcal{X}$; a metric on distributions, $\mathcal{M}(\mathcal{X})$, can be defined

$$\mathcal{W}_p(\alpha, \beta) = \mathcal{L}_{d^p}(\alpha, \beta)^{1/p}.$$

This is known as the p-Wasserstein distance and has become widely used for comparison between densities where the supports do not overlap (e.g. in image interpolation [27] and machine learning [52]). In particular $\mathcal{W}_2$ has nice properties, which allow translations to be factored out. The Wasserstein distances metrize weak convergence. That is, on a compact domain, a sequence (α_k) converges weakly to $\alpha \in \mathcal{M}(\mathcal{X})$ if and only if $\mathcal{W}_p(\alpha_k, \alpha) \to 0$ as $k \to \infty$ [182].

In Figure 5.2, we plot the first two terms of the sequence of measures $\alpha_n = \delta_{1-1/n}$. Popular information divergences, like the Kullback–Leibler (KL) divergence, will give a divergence of infinity between α_n and $\alpha_\infty = \delta_1$ as the supports do not overlap. Wasserstein distances will capture this convergence as n increases since the Wasserstein distance between α_n and α_∞ is $1/n$.

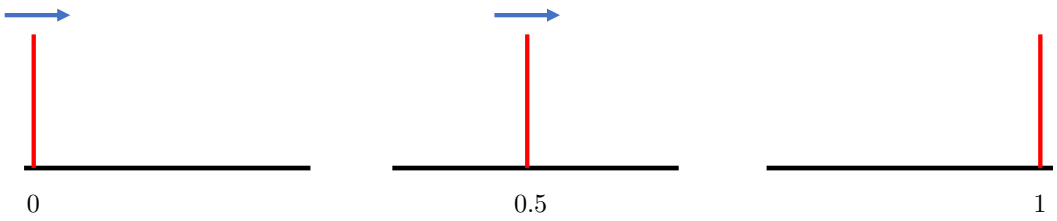


Figure 5.2: Example measures from the family of dirac delta measures $\alpha_n = \delta_{1-1/n}$ for $n = 1$ (left), and $n = 2$ (middle). The plot on the right is $\alpha_\infty = \delta_1$. The KL divergence between any of these is ∞ but the Wasserstein distance between α_n and α_∞ is $1/n$. The blue arrow indicates the direction in which the measure moves with increasing n.

In the case $c(x, y) = ||x - y||^2$, provided the densities are not singular, Brenier's theorem [29] tells us that the Monge Problem and the Kantorovich relaxation are equivalent. Brenier's results were extended to strictly convex costs in [73].

As the Kantorovich relaxation is a linear program, we can formulate its dual problem by manipulating the Lagrangian of the original problem. Since the primal is a constrained minimsation problem, its dual is a constrained maximisation problem

$$L_C(\boldsymbol{a}, \boldsymbol{b}) = \max_{(\boldsymbol{f},\boldsymbol{g})\in R(C)} \big\{\langle \boldsymbol{f}, \boldsymbol{a}\rangle + \langle \boldsymbol{g}, \boldsymbol{b}\rangle\big\},$$

where $R(C) := \{(\boldsymbol{f}, \boldsymbol{g}) \in \mathbb{R}^m \times \mathbb{R}^n : (\boldsymbol{f}\mathbb{1}_n^\intercal + \mathbb{1}_m \boldsymbol{g}^\intercal)_{i,j} \leq C_{i,j} \ \forall i, j\}$. The dual variables $\boldsymbol{f}, \boldsymbol{g}$ are known as *Kantorovich potentials*.

5.2.2 The regularised problem

By regularising the Kantorovich problem, we can gain computational benefits. Let the discrete entropy, $H(P)$, be given by

$$H(P) := -\sum_{i,j} P_{i,j}(\log(P_{i,j}) - 1) = -\langle P, \log P - \mathbb{1}_{n\times m}\rangle, \tag{5.7}$$

taking the convention $H(P) = -\infty$ if any $P_{i,j} < 0$, and $P_{i,j} \log P_{i,j} = 0$ for $P_{i,j} = 0$. Note that the entropy (5.7) has been translated compared to its usual form, in order to obtain a more convenient form of the derivative to work with.

We define the regularised problem as

$$L_C^\epsilon(\boldsymbol{a}, \boldsymbol{b}) := \min_{P\in U(\boldsymbol{a},\boldsymbol{b})} \big\{\langle P, C\rangle - \epsilon H(P)\big\}, \tag{5.8}$$

where $H(P)$ is the discrete entropy of the coupling matrix P, and $U(\boldsymbol{a}, \boldsymbol{b})$ is as in (5.5). We now see that this regularised problem is numerically tractable.

Note that this choice of regulariser is strongly concave (the Hessian $-\mathrm{diag}(1/P_{i,j})$ is negative definite). Therefore the regularised problem has a strongly convex objective and hence has a unique solution. This unique solution is the one that minimises the KL divergence, and corresponds to the Gibbs kernel [141]

$$K_{i,j} = \exp\{-C_{i,j}/\epsilon\}. \tag{5.9}$$

For small ϵ this problem resembles the unregularised optimal transport problem. For larger values of ϵ, the solution resembles the case where the marginal distributions are 'independent', i.e. $P_{i,j} = a_i b_j$. Larger values of ϵ also results in faster convergence of algorithms. Cominetti and San Martín [50] showed that, provided that the original problem has a nonempty and bounded optimal solution set, the solution of this regularised problem converges to a solution of the unregularised problem as $\epsilon \to 0$.

The Lagrangian of the regularised problem (5.8) is given by

$$\mathscr{L}(P, \boldsymbol{f}, \boldsymbol{g}) = \langle P, C\rangle - \epsilon H(P) - \langle \boldsymbol{f}, P\mathbb{1}_n - \boldsymbol{a}\rangle - \langle \boldsymbol{g}, P^\intercal \mathbb{1}_m - \boldsymbol{b}\rangle. \tag{5.10}$$

By first order optimality conditions, we have

$$\frac{\partial \mathscr{L}(P, \boldsymbol{f}, \boldsymbol{g})}{\partial P_{i,j}} = C_{i,j} - \epsilon \frac{\partial H(P)}{\partial P_{i,j}} - f_i - g_j = 0.$$

Now

$$\frac{\partial H(P)}{\partial P_{i,j}} = -\frac{\partial}{\partial P_{i,j}}(P_{i,j}(\log(P_{i,j}) - 1)) = -\log(P_{i,j}),$$

so

$$P^*_{i,j} = \exp\{f_i/\epsilon\}\exp\{-C_{i,j}/\epsilon\}\exp\{g_j/\epsilon\}. \tag{5.11}$$

Let us define $\boldsymbol{u} \in \mathbb{R}^m$ such that $u_i = \exp\{f_i/\epsilon\}$, $\boldsymbol{v} \in \mathbb{R}^n$ such that $v_j = \exp\{g_j/\epsilon\}$ and K as in (5.9). We can write the optimal transport plan (5.11) as

$$P^* = \text{diag}(\boldsymbol{u})K\text{diag}(\boldsymbol{v}), \tag{5.12}$$

where $\text{diag}(\boldsymbol{u})$ is the diagonal matrix with elements of $\boldsymbol{u}$ along the diagonal. Since P needs to conserve mass, i.e. $P \in U(\boldsymbol{a}, \boldsymbol{b})$, hence

$$\text{diag}(\boldsymbol{u})K\text{diag}(\boldsymbol{v})\mathbb{1}_n = \boldsymbol{a}, \quad \text{diag}(\boldsymbol{v})K^\intercal\text{diag}(\boldsymbol{u})\mathbb{1}_m = \boldsymbol{b}.$$

This can be rewritten in terms of elementwise multiplication, $\odot$,

$$\boldsymbol{u} \odot (K\boldsymbol{v}) = \boldsymbol{a}, \quad \boldsymbol{v} \odot (K^\intercal\boldsymbol{u}) = \boldsymbol{b}.$$

Therefore this is a matrix scaling problem and we can apply the Sinkhorn algorithm [168, 57], Algorithm 5, to find $\boldsymbol{u}$ and $\boldsymbol{v}$.

Algorithm 5: Sinkhorn Algorithm

Input $K = \exp\{-C/\epsilon\}$
Initialise $\boldsymbol{v}^{(0)} = \mathbb{1}_n$
while $|\boldsymbol{v} \odot (K^\intercal\boldsymbol{u}) - \boldsymbol{b}|_\infty >$ *tolerance* **do**
  $\boldsymbol{u}^{(k+1)} = \boldsymbol{a} \oslash K\boldsymbol{v}^{(k)}$, where $\oslash$ denotes elementwise division
  $\boldsymbol{v}^{(k+1)} = \boldsymbol{b} \oslash K^\intercal\boldsymbol{u}^{(k+1)}$
end

These iterations converge [71], however different initialisations will lead to different solutions for $\boldsymbol{u}$ and $\boldsymbol{v}$, they are only defined up to a multiplicative constant. That is, $\boldsymbol{u}$, $\boldsymbol{v}$ are solutions imply $\lambda\boldsymbol{u}, 1/\lambda\boldsymbol{v}$ are also solutions, $\lambda > 0$. However, the optimal coupling, P^*, is unique and can be recovered by (5.12).

Numerical stability deteriorates as $\epsilon \to 0$, in particular, the Sinkhorn algorithm will fail to converge as soon as elements in K become negligible since this will result in division by 0. In order to approximate unregularised problems, decaying schedules for ϵ have been considered, combined with multigrid methods (Section 2.3, [161]). Note that we can use other strongly convex regularisers and the theory above will follow (see e.g. [62]).

We can substitute the optimal P (5.11) back into the Lagrangian (5.10) (noting that the Lagrange multiplier terms vanish)

$$\begin{aligned}
&\langle P,\ C\rangle + \epsilon\left\langle P,\ \log\left(\text{diag}\left(e^{\boldsymbol{f}/\epsilon}\right)e^{-C/\epsilon}\text{diag}\left(e^{\boldsymbol{g}/\epsilon}\right)\right) - \mathbb{1}_{n\times m}\right\rangle \\
&= \langle P,\ C\rangle + \left\langle P,\ \boldsymbol{f}\mathbb{1}_n^T - C + \mathbb{1}_m\boldsymbol{g}^\intercal - \epsilon\mathbb{1}_{n\times m}\right\rangle \\
&= \left\langle P,\ \boldsymbol{f}\mathbb{1}_n^T\right\rangle + \left\langle P,\ \mathbb{1}_m\boldsymbol{g}^\intercal\right\rangle - \epsilon\left\langle P,\ \mathbb{1}_{n\times m}\right\rangle.
\end{aligned}$$

We can simplify these terms further

$$\left\langle P,\ \boldsymbol{f}\mathbb{1}_n^T\right\rangle = \sum_{i,j} P_{i,j}f_i\mathbb{1}_j = \sum_{i,j} f_iP_{i,j}\mathbb{1}_j = \sum_i f_ia_i = \langle \boldsymbol{f},\ \boldsymbol{a}\rangle,$$

$$\langle P,\ \mathbb{1}_m \boldsymbol{g}^\intercal \rangle = \sum_{i,j} P_{i,j} \mathbb{1}_i g_j = \sum_{i,j} g_j P_{i,j} \mathbb{1}_i = \sum_i g_j b_j = \langle \boldsymbol{g},\ \boldsymbol{b} \rangle \,,$$

$$\begin{aligned} \langle P,\ \mathbb{1}_{n\times m} \rangle &= \left\langle \operatorname{diag}\big(e^{\boldsymbol{f}/\epsilon}\big) e^{-C/\epsilon} \operatorname{diag}\big(e^{\boldsymbol{g}/\epsilon}\big),\ \mathbb{1}_{n\times m} \right\rangle \\ &= \sum_{i,j} \exp\{f_i/\epsilon\} \exp\{-C_{i,j}/\epsilon\} \exp\{g_j/\epsilon\} \\ &= \left\langle e^{\boldsymbol{f}/\epsilon}, K e^{\boldsymbol{g}/\epsilon} \right\rangle . \end{aligned}$$

Hence we have

$$L_C^\epsilon(\boldsymbol{a}, \boldsymbol{b}) = \max_{\boldsymbol{f}\in\mathbb{R}^m, \boldsymbol{g}\in\mathbb{R}^n} \langle \boldsymbol{f},\ \boldsymbol{a} \rangle + \langle \boldsymbol{g},\ \boldsymbol{b} \rangle + \epsilon \big\langle e^{\boldsymbol{f}/\epsilon}, K e^{\boldsymbol{g}/\epsilon} \big\rangle. \tag{5.13}$$

Therefore $L_C^\epsilon(\boldsymbol{a}, \boldsymbol{b})$ is linear, and hence convex, in both $\boldsymbol{a}$ and $\boldsymbol{b}$. We can find the gradient of the loss function

$$\nabla L_C^\epsilon(\boldsymbol{a}, \boldsymbol{b}) = [\boldsymbol{f}^*, \boldsymbol{g}^*] \tag{5.14}$$

where $\boldsymbol{f}^*, \boldsymbol{g}^*$ are the optimisers of (5.13), such that their coordinates sum to zero to preserve mass. This strategy is utilised in variational optimal transport problems (in combination with some suitable discretisation scheme).

Remark 5.2.1. *In general applications, the optimal transport problem as described may only form part of the picture. In particular, the distribution that we are mapping the initial distribution to may not be fixed. Variational optimal transport is a class of problems where we search over possible distributions to map from or to, and find the lowest transport cost in each case [62, 133]. This is the case, for example, in learning a dictionary/centroid for clustering, the source measure is a set of points that would minimise the Wasserstein distance but the location of these are not pre-defined.*

5.3 Continuous Optimisation

Continuous optimisation underpins the machine learning methods we looked at in Part I. In this section, we summarise the basics of continuous optimisation and specific techniques that we rely upon in this second part of the thesis.

5.3.1 Introduction

Optimisation problems can be partitioned as constrained and unconstrained problems. In unconstrained problems, there is some real-valued objective function, $f : \mathbb{R}^n \to \mathbb{R}$, to be minimised (or equivalently maximised). In constrained problems, we impose further conditions on the domain of f, for example, bounds or positivity.

Typical continuous optimization methods generate a sequence of iterates x_k that converge to the solution from some user-defined starting point x_0. A canonical continuous optimization method is steepest descent, this is a line search method where the search direction is given by $-\nabla f$. For a twice continuously differentiable function with exact line search which converges to some point x^*, it can be shown that for sufficiently large iterations k,

$$f(x_{k+1}) - f(x^*) \leq r^2 [f(x_k) - f(x^*)] \quad \text{for some } r \in \mathbb{R}.$$

For further background, see [137].

For non-convex continuous optimisation, naïve gradient methods in general may fail to converge to a local optimum, let alone a global optimum [134]. In high dimensions, second-order methods (even quasi-Newton) are prohibitively expensive, as computing the Hessian is $O(n^2)$. Quasi-Newton methods require computing and storing a history of all $O(n)$ gradients encountered and so may also be too expensive.

Even in the case of convex problems, there are no guarantees that gradient methods will converge. For example, optimising a quadratic cost function with the wrong step size can lead to cycling [146]. Although backtracking linesearch would ensure convergence on convex quadratic problems, linesearch in high dimensions can also be prohibitively expensive, as linesearch inherently requires repeated evaluations of the objective function at different candidate step sizes to check if the step ensures sufficient decrease in the objective.

For a linear objective if there are nonlinear, and in particular nonconvex, constraints, gradient methods may still fail to converge to the desired optimum. This is because such problems are often highly multimodal and continuous optimization primarily focuses on practically finding *local* rather than *global* minima.

5.3.2 Gradient projection methods

For constrained optimisation problems, we may want to take an arbitrary step, and then apply a projection so that the solution is feasible after each iteration.

The Euclidean projection of y onto a set X is defined as

$$\Pi_X(y) = \arg\min_{x \in X} ||x - y||_2^2. \tag{5.15}$$

The projected gradient descent (PGD) method has updates of the form

$$x_{k+1} = \Pi_X(x_k - \nabla f(x_k)), \quad \forall k \geq 1, \tag{5.16}$$

to ensure that solutions are within some constrained space X. Note that each iterate in the PGD can be viewed as a minimiser of the quadratic approximation of the objective function

$$x_{k+1} = \arg\min_{x \in X} \left\{ f(x_k) + \langle \nabla f(x_k), x - x_k \rangle + \frac{\mu}{2} ||x - x_k||^2 \right\}. \tag{5.17}$$

The convergence of projected gradient methods for linearly constrained problems was shown by Calamai and Moré [34].

An efficient way to project to the simplex

For certain types of sets X, there are efficient algorithms for the projection to minimise the Euclidean distance. We look at the case where we want to find the projection of an arbitrary point in $\mathbb{R}^D$ onto the simplex

$$\sum_{i=1}^{D} a_i = n, \quad a_i \geq 0, \ \forall i. \tag{5.18}$$

Since this is a quadratic problem and the objective is strictly convex, there is a unique optimal point for the projection. A direct method of order $\mathcal{O}(D \log(D))$ [187] is replicated in Algorithm 6. The intuition of this algorithm is that each element in y is collapsed onto a 1-D axis, the points are then shifted by λ such that the points on the positive side of the axis will add up to the correct total of n.

Algorithm 6: Euclidean projection of a vector onto the simplex (5.18)

input : $y \in \mathbb{R}^D$
Sort the elements of y into ascending order and call this sorted vector u, i.e. u is a permutation of y such that $u_1 \geq u_2 \geq \cdots \geq u_D$.
Find $\rho = \max\left\{1 \leq j \leq D : u_j + \frac{1}{j}\left(n - \sum_{i=1^j} u_i\right) > 0\right\}$
Set $\lambda = \frac{1}{\rho}\left(n - \sum_{i=1}^{\rho} u_i\right)$
output: x, such that $x_i = \max\{y_i + \lambda, 0\}, i = 1, ..., D$.

5.3.3 Coordinate descent methods

Next we look at *coordinate descent methods* [192], which are iterative methods. At each iteration in coordinate descent, one of the coordinate directions is used to search for a descent step (a simpler, lower dimension optimisation problem)

$$\min_a f(x_1, x_2, \ldots, a, \ldots, x_n).$$

The similarity to a Gauss–Seidel update means that this may also be referred to as the *nonlinear Gauss–Seidel method*.

In *block* coordinate descent [192], a subset of the coordinates is used to find a descent step. How to choose the sequence of coordinates/blocks of coordinates to optimise over is a design choice. A simple, naïve approach is to cycle through all the coordinates in order [137]. Other choices include symmetric forwards and backwards sweeps of the coordinates [12] and choosing the direction with the largest gradient value for some fixed step-size (Gauss–Southwell rule) [170].

Failure to converge may be caused by cycling. Coordinate descent for convex, differentiable functions can still lead to cycling [146].

The convergence rate of coordinate descent methods is slower compared with steepest descent, but the advantage is that it can be used in a derivative-free setting as well. Modifications and specific search schedules have been proposed for better theoretical convergence properties, e.g. global convergence [137].

Nonconvex coordinate descent methods

While there is existing literature on coordinate descent algorithms for nonconvex objectives, these works usually assume some particular type of structure on the objective function. For example [140] considers a class of objective functions, which are both nonconvex and non-smooth but can be decomposed into two parts, $F(x) = f(x) + h(x)$, where $f(x)$ is smooth and nonconvex, and $h(x)$ is (non-differentiable) continuous, block separable, and convex. Let

$$F^* = \min_{x \in \mathbb{R}^n} F(x) = \min_{x \in \mathbb{R}^n} \left\{f(x) + h(x)\right\}.$$

The authors prove that if the blocks are uniformly sampled, then there is asymptotic convergence to stationary points, that is, any accumulation point is a stationary point.

For an unconstrained problem, only one block is being changed at a time whilst for the (singly) linearly constrained case (by further imposing that $a^\intercal x = b$, two independent blocks are being changed. Note that [140] states that their algorithm would generalise for a greater number of linear constraints, but the complexity of each step of the algorithm would increase.

The authors assume that each block in the gradient of f satisfies a Lipschitz condition, to be precise, for the ith block (which has dimension n_i), we have

$$||\nabla_i f(x + U_i s_i) - \nabla_i f(x)|| \leq L_i ||s_i|| \quad \forall s_i \in \mathbb{R}^n, x \in \mathbb{R}^n,$$

where U_i is a notational convenience to map the perturbation of s_i in $\mathbb{R}^{n_i}$ to the whole space in $\mathbb{R}^n$. With the Lipschitz condition, the authors obtain an upper bound for $F(x + U_i s_i)$ (see [140] for details)

$$F(x + U_i s_i) \leq f(x) + \langle \nabla_i f(x), s_i \rangle + \frac{L_i}{2} ||s_i||^2 + h(x + U_i s_i) \quad \forall s_i \in \mathbb{R}^{n_i}, x \in \mathbb{R}^n.$$

Then each step of the algorithm finds the direction that would minimise this upper bound

$$\min_{s_i \in \mathbb{R}^{n_i}} f(x) + \langle \nabla_i f(x), s_i \rangle + \frac{L_i}{2} ||s_i||^2 + h(x + U_i s_i).$$

Other formulation of block coordinate descent methods also exist. In [194] a very similar objective value function form is assumed, except h is no longer required to be convex, just proximable, which means the proximal operator, $\arg\min_x \{h(x) + \frac{1}{2}||x - y||^2\}$, can be easily obtained. Note that this is a generalisation of a projection operator; the proposed method generalises proximal gradient descent and the authors show global convergence under the Kurdyka–Łojasiewicz property.

Chapter 6

An optimal transport inspired approach to Air Traffic Control

In this chapter, we approach the arrival management problem as an entropy regularised optimisation problem, inspired by optimal transport theory. Drawing on the Kantorovich approach to optimal transport, we consider the distribution of arrival times of each aircraft, and seek to optimize this collection of distributions.

Our optimisation problem involves reducing a cost that represents the interaction between aircraft which are too close together. This 'conflict' cost can be viewed as a quadratic term, but since each aircraft does not interact with itself, there are zeros along the diagonal, hence the corresponding quadratic term is not positive definite. As a result, our objective function is not convex. We introduce a block coordinate descent-like algorithm, by optimising each row (corresponding to one aircraft) separately. This significantly reduces the dimension of the optimisation problem.

Note that for the rest of the chapter, we will use the `python` convention of zero-based indexing.

6.1 Air Traffic Control Motivation

We can cast the arrival management problem as a regularised optimal transport-style problem where the target measure is not fixed. There are a few ways we can set up our problem.

6.1.1 Monge Formulation

Let us start by discussing a Monge formulation of the problem. For simplicity, let us begin with only one aircraft size. Suppose we are given a list of normalised predicted arrival times $t = [t_0, ..., t_{n-1}] \in [0, 1]^n$. We discretise $[0, 1]$ into N intervals. Given some fixed separation gap needed between successive aircraft, this will not result in a significant loss of accuracy, provided we choose a sufficiently large N. Let the minimum separation gap needed be q intervals/slots in our discretised grid.

We map the list of normalised arrival times into a frequency vector. For clarity, if we are given the normalised arrival times $t = [0.11, 0.32, 0.67]$, and we have a discretisation of $N = 10$ spaces, then t would be mapped to $[0, 1, 0, 1, 0, 0, 1, 0, 0, 0]$ as the frequency vector. This gives us the initial distribution μ in the form of (5.2). For the Monge set-up, as particle splitting (as seen in Figure 5.1) is not allowed, it is important to choose N large enough

that when we map from the normalised arrival times to the frequency vector, there is at most one aircraft per slot (otherwise the overlapping aircraft will always be kept together under any Monge formulation). We want to minimise the number of conflicts. Let us define a conflict cost as a (nonlinear) function on the frequency vector, $R : \{0,1\}^N \to \mathbb{R}$. There is a pre-defined (per domain knowledge) transport cost function for changing the slot of each aircraft, given by the cost matrix C, so c_{ij} represents the cost of moving an aircraft from slot i to slot j.

In the case where there are conflicts to be resolved, we want to map our initial frequency vector μ onto another frequency vector ν (representing the target frequency vector) to reduce/remove the conflicts whilst keeping the effort required (transport cost) to change configurations low. That is, we want to map the starting distribution onto another feasible distribution to minimise a weighted sum of the conflict cost and the transport cost.

Since we require that ν only has unit masses, mapping to an alternative frequency distribution can be thought of as an optimal assignment problem, with the transport cost c_{ij} being the value of object j to bidder i. Assignment problems, and more generally minimum cost flow problems, have been studied extensively, with popular methods including the Hungarian method [111] and auctions [22, 23]. These generally are combinatorially complex, and hence may be unsuitable for large problems. In our problem, we need to place further constraints due to the complication of minimum separation gaps and conflict cost. If we think of planes as bidders, this means that once a bidder has made a purchase, a group of goods (slots) become unavailable/more expensive to others. Essentially we require that a bidder buys a group of items, that is, a block of time slots. This resembles combinatorial auction designs. Each bidder can give what they believe is the fair price for some block of space in a setting like the Vickrey auction [181]. However, finding the right split, or the Shapley value in coalitions, are NP-hard problems. There has been recent work to generalise this to continuous space and apply auction design ideas to optimal transport problems [185], but the applicability is limited.

6.1.2 Kantorovich Formulation

As we do not want to be confined to this discrete NP-hard setting, and we have seen the benefits regularisation can bring, let us proceed to use the Kantorovich problem formulation instead.

Vector optimisation

Assuming the same notation as in Section 6.1.1, now μ and ν are not restricted to unit values. To ensure that everything is well-behaved, we can formulate this as the regularised problem

$$\min_{\nu}\{\alpha R(\nu) + \beta L_C^{\epsilon}(\mu,\nu)\}, \tag{6.1}$$

where α and β are parameters, and

$$L_C^{\epsilon}(\mu,\nu) = \min_{P\in U(\mu,\nu)} \big\{\langle C, P\rangle - \epsilon H(P)\big\}, \tag{6.2}$$

where $H(P)$ is the entropy as defined by (5.7).

Define the ith element of the discrete convolution as

$$(\mu * \nu)_i = \sum_{m=-\infty}^{\infty} \mu_m \nu_{i-m+1}. \tag{6.3}$$

We can express the conflict cost, R, as a convolution of the form

$$\begin{aligned} R(\nu) &= \left(\sum_{i=1}^{N} \nu_i((\nu * \mathbb{1}_q)_i) \right) - \nu^\intercal \mathbb{1}_N \\ &= \left(\sum_{i=1}^{N} \nu_i \left[\sum_{m=-\infty}^{\infty} \nu_m (\mathbb{1}_q)_{i-m+1} \right] \right) - \nu^\intercal \mathbb{1}_N \\ &= \left(\sum_{i=1}^{N} \nu_i \left[\sum_{m=\max(1,i+1-q)}^{i} \nu_m \right] \right) - \nu^\intercal \mathbb{1}_N, \end{aligned}$$

which counts the number of overlaps each occupied slot has with subsequent occupied slot within the minimum separation distance (given by q). Note that the last term $\nu^\intercal \mathbb{1}_N$ removes any overcounting based on self-interactions.

As discussed in Section 5.2.2, from the theory of entropic regularised optimal transport, we can find the analytic optimisers of the loss function via Kantorovich potentials. At each step we are still trying to solve two optimisation problems, and a naïve gradient step would not preserve the total mass of ν or respect the positivity constraint, therefore this method will have to be used in combination with e.g. gradient projection methods, as discussed in Section 5.3.2. Note that our cost is nonlinear in the measure ν (that is, due to the conflict cost term, it is not a simple transport cost on the space of measures).

Matrix optimisation

Let us look at optimising the transport matrix P directly. The transport matrix tells us how much mass is moved from one location to another. That is, p_{ij} gives the amount of mass moved from the ith location to the jth location. We want to penalise planes getting sent to the same location, which is a penalty on the columns. The matrix P is stochastic in the rows, that is, each row is in the probability simplex $\mathcal{S}_N$. However, we have no fixed target distribution so as a result, our columns are not similarly constrained. This is essentially a variational optimal transport problem as discussed in Section 5.2.

Figure 6.1 gives some intuition on what we want the conflict cost to penalise in this set-up. Pictorially, each row of the transport matrix represents the measure of the location of one aircraft (and so lives on the probability simplex S_N). If we take a convolution of the aircraft locations against the minimum separation gap, we find all slots which are affected by the turbulence in the air. Then we can sum the columnwise interactions for all affected spaces (represented by the airplane and swirl icons in the diagram).

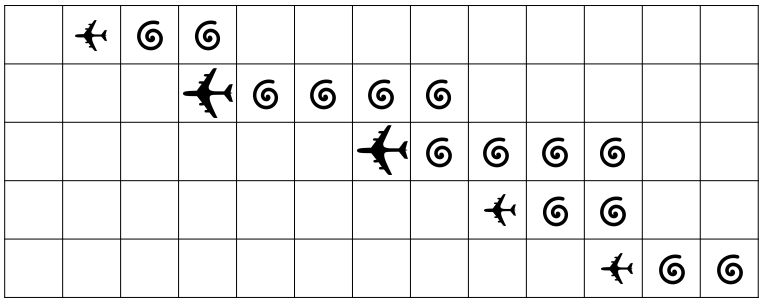


Figure 6.1: A pictorial representation of the convolved transport matrix.

Formally, as we are now optimising over the transport matrix, we have a greater number of degrees of freedom, so let us redefine the conflict cost to be a function on P instead, so that $R : \mathcal{S}_N^n \to \mathbb{R}$. Recall that q is the required separation gap between successive aircraft, we start by finding the convolution (6.3) against $Q = \mathbb{1}_q$ (the vector of ones of length q) along each of the rows of the transport matrix, which we denote $\hat{P}$. This means that all columns which the wake vortex of the aircraft impacts are populated (as represented by the swirls in Figure 6.1). Then sum the pairwise product along the columns, excluding self interactions, i.e. $i \neq j$ in the sum for (6.4). That is, let us take each row of $\hat{P}$ to be

$$\begin{aligned}\hat{P}_{i,\cdot} =& P_{i,\cdot} * Q,\\ \hat{P}_{ij} =& \sum_{m=-\infty}^{\infty} P_{im} Q_{j-m+1}\\ =& \sum_{m=\max(1,j-q+1)}^{\min(j,N)} P_{im},\end{aligned}$$

where $*$ denotes the convolution (as defined in (6.3)). Then we compute the number of conflicts as

$$\begin{aligned}R(P) =& \sum_{i\neq j}\sum_{k} \hat{P}_{ik}\hat{P}_{jk}\\ =& \sum_{i\neq j}\sum_{k}\left(\sum_{m=\max(1,k-q+1)}^{\min(k,N)} P_{im}\right)\left(\sum_{m=\max(1,k-q+1)}^{\min(k,N)} P_{jm}\right).\end{aligned} \tag{6.4}$$

Hence our problem is now

$$\min_P \left\{\alpha R(P) + \beta L(P) - \epsilon H(P)\right\}, \tag{6.5}$$

where $L(P) = \langle P, C\rangle$ and $H(P)$ is the entropy as defined by (5.7).

Note that whilst the conflict cost, R, can be viewed as a quadratic term, since each aircraft does not interact with itself, there are zeros along the diagonal, so this is not a positive definite matrix and hence R is not convex. The regulariser, H, is $\epsilon-$strongly convex and L is linear. This means that the only nonconvexity in the objective function is coming in through the conflict costs. As the conflict cost introduces non-convexity to the problem, we shall see that increasing the parameter α to penalise conflicts can create problems for the optimisation.

Quadratic programming formulation

Our formulation of the conflict cost (6.4) can be rewritten (in block QP form in the more general case of multiple plane types/weight-based separation)

$$R(p) = p^\intercal B p = \begin{pmatrix} p_0^\intercal & p_1^\intercal & \dots & p_{n-1}^\intercal\end{pmatrix}\begin{pmatrix} 0 & A_{0,1} & \dots & A_{0,n-1}\\ A_{1,0} & 0 & \dots & A_{1,n-1}\\ \vdots & \vdots & \ddots & \vdots\\ A_{n-1,0} & \dots & \dots & 0\end{pmatrix}\begin{pmatrix} p_0\\ p_1\\ \vdots\\ p_{n-1}\end{pmatrix},$$

where p_i denotes the ith row of the transport matrix, A_{ij} denotes the matrix formed by the convolutions between separation gaps for type (weight category) of plane i and plane j (see example below), and 0 represents the $n \times n$ zero matrix.

As an example, if plane i's separation gap is 2 and plane j's separation gap is 4, then we obtain

$$A_{ij} = \begin{pmatrix} 2 & 1 & 0 & 0 & 0 & 0 \\ 2 & 2 & 1 & 0 & 0 & 0 \\ 2 & 2 & 2 & 1 & 0 & 0 \\ 1 & 2 & 2 & 2 & 1 & 0 \\ 0 & 1 & 2 & 2 & 2 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{pmatrix},$$

where the coefficient is obtained by convolving and truncating $\mathbb{1}_2$, and $\mathbb{1}_4$. There is an edge case at the end where $\mathbb{1}_1$, and $\mathbb{1}_4$ are convolved.

Omitting regularisation, our problem then becomes

$$\min_{p \in \mathcal{S}_n} \left\{ p^\intercal B p + L(p) \right\}.$$

Unfortunately, as mentioned above, this does not form a convex QP problem since the matrix B is not positive definite. This is also a very high dimensional problem. As a result, there are limited optimisation tools available. IPOPT [183] is a notable exception however its use has not been explored in this thesis.

We can try to take an arbitrary gradient step then use gradient projection methods as discussion in Section 5.3.2 to map the next iterate back into the solution space. However, for high dimension simplices, the projection most likely maps to a vertex, making gradual steps infeasible. This is something we have implemented and it does not perform well in practice. As such, we will not be discussing this approach and the results in any more detail in this thesis.

6.2 A row-wise gradient algorithm

By adding an entropic regulariser to the optimisation problem, we can optimise subsets of the problem exactly. We optimise each aircraft separately, and so design a block coordinate descent-like algorithm for our entropy regularised problem (6.5). We show in Section 7.2 that this is locally Pareto optimal, but it is not a social optimum.

6.2.1 Global gradient descent

A naive gradient step would be to update the transport matrix as

$$P \leftarrow P - \gamma \left(\alpha \frac{\partial R}{\partial P} + \beta \frac{\partial L}{\partial P} - \epsilon \frac{\partial H}{\partial P} \right).$$

We find

$$\begin{aligned}\frac{\partial L}{\partial P_{ij}} =& C_{ij},\\ \frac{\partial R}{\partial P_{ik}} =& \sum_{j\neq i} \sum_{m=\max(1,k-q+1)}^{k+q-1} (q-|k-m|)p_{jm}, \quad 0 \leq k \leq N-q,\\ \frac{\partial R}{\partial P_{ik}} =& \sum_{j\neq i} (\mathbb{1}_{N-k+1} * \mathbb{1}_q) \cdot p_{j,k-q+1:\min(k+q-1,N)}, \quad k > N-q.\end{aligned}$$

For multiple aircraft types, there will be different separation gaps associated with each type of aircraft. Let the minimum separation distance of the ith aircraft be denoted $q(i)$. Note that this is a simplification, as here only the leader is considered when defining acceptable gaps. We have to convolve each row with the relevant gap size $Q(i) = \mathbb{1}_{q(i)}$, and so in the general case we have

$$\frac{\partial R}{\partial P_{ik}} = \sum_{j\neq i} (\mathbb{1}_{q(i)} * \mathbb{1}_{q(j)})_{[-(q(i)+k):]} \cdot p_{j,\max(1,k-q(j)+1):k+q(i)-1}, \quad k < q(i)-1, \tag{6.6}$$

$$\frac{\partial R}{\partial P_{ik}} = \sum_{j\neq i} (\mathbb{1}_{q(i)} * \mathbb{1}_{q(j)}) \cdot p_{j,\max(1,k-q(j)+1):k+q(i)-1}, \quad q(i)-1 \leq k \leq N-q(i), \tag{6.7}$$

$$\frac{\partial R}{\partial P_{ik}} = \sum_{j\neq i} (\mathbb{1}_{N-k+1} * \mathbb{1}_{q(j)}) \cdot p_{j,k-q(j)+1:\min(k+q(i)-1,N)}, \quad k > N-q(i). \tag{6.8}$$

Note that the index on $\mathbb{1}_{q(i)} * \mathbb{1}_{q(j)}$ in (6.6) ensures that the vector is the same length as the p vector for the dot product. A negative index means to take the last $q(i)+k$ elements.

Since taking this step will almost certainly destroy the stochastic matrix property of the transport matrix, we can perform an inner optimisation to find the best direction that would preserve the stochastic matrix property. For each row of the gradient, we are looking for the direction that would be most aligned with the computed gradient, which ensures positivity of the transport matrix after the update and that the elements of each row still sum to 1. However, satisfying the constraints would involve the intersection of many half planes, and would be costly to optimise.

6.2.2 Row-wise gradient steps

Observe that if we only iteratively update one row at a time and assume all others are fixed, we can just absorb the conflict cost as part of the "transport cost", this gives a linear (hence convex) marginal cost function for a single row. Row-wise optimisation also lowers the dimension of our optimisation problem which is convenient in practice.

Given a transport matrix P, let us recall our objective function

$$\min_P \big\{ R(P) + \langle C, P\rangle + \epsilon\langle P,\, \log P - \mathbb{1}_{n\times N}\rangle \big\}. \tag{6.9}$$

The term $R(P)$ represents the number of conflicts, which is a linear combination of the entries in the other rows. Note that the contribution of each row to the cost is also a linear combination of elements of each row, with the coefficient determined by convolutions. Therefore the row-wise problem for the the ith row can be written as

$$\min_{\boldsymbol{p}} \ \big\{ \boldsymbol{k_i}^\intercal \boldsymbol{p} + \epsilon \boldsymbol{p}^\intercal (\log(\boldsymbol{p}) - \mathbb{1}_n) \big\}.$$

The jth element of $\boldsymbol{k_i}$ is given by

$$k_{ij} = \partial R/\partial P_{ij} + C_{ij}. \tag{6.10}$$

We can think about this iterative update as solving the following constrained minimisation problem

$$\begin{aligned} \min_{\boldsymbol{p}} \ & \left\{\boldsymbol{k_i}^\intercal \boldsymbol{p} + \epsilon \boldsymbol{p}^\intercal(\log(\boldsymbol{p}) - \mathbb{1}_n)\right\} \\ \text{such that } & \mathbb{1}_n^\intercal \boldsymbol{p} = 1 \\ & \boldsymbol{p} \geq 0. \end{aligned}$$

Let us set aside the inequality condition for the moment, then we can use Lagrange multipliers to solve the unconstrained problem

$$\min_{\boldsymbol{p}} \ \left\{\boldsymbol{k_i}^\intercal \boldsymbol{p} + \epsilon \boldsymbol{p}^\intercal(\log(\boldsymbol{p}) - \mathbb{1}_n) - \lambda(\mathbb{1}_n^\intercal \boldsymbol{p} - 1)\right\}.$$

First order optimality conditions give (along with the the constraint $\mathbb{1}_n^\intercal \boldsymbol{p} - 1 = 0$):

$$\boldsymbol{k_i} + \epsilon \log(\boldsymbol{p}) + \lambda \mathbb{1}_n = 0,$$

from which we obtain $\lambda = -\log(\mathbb{1}_n^\intercal \exp\{\boldsymbol{k_i}/\epsilon\})$ and so the solution to the optimisation problem is given by

$$\boldsymbol{p}^* = \frac{\exp\{-\boldsymbol{k_i}/\epsilon\}}{\mathbb{1}_n^\intercal \exp\{-\boldsymbol{k_i}/\epsilon\}}. \tag{6.11}$$

The optimal solution of the problem is in exponential form and therefore satisfies the positivity constraint automatically. So that the algorithm is more stable, we can then update the relevant row by taking a weighted average between the current value and $\boldsymbol{p}^*$. This is analogous to the relaxation factor used in iterative numerical methods (for both linear and nonlinear systems). The complete method can be seen in Algorithm 7.

Let us test this algorithm on a couple of simple examples. For simplicity, let us use a cubic cost for the transport cost, that is, the entries of our cost matrix are $C_{ij} = |c(i-j)|^3$ for some scalar c. We have used this symmetric superquadratic cost so that it is only viable for aircraft to move a few intervals away.

In Figures 6.2 and 6.3 we see the results for an example where $c = 1$, $\alpha = 40$, $\beta = 1$, $\epsilon = 1$, and the minimum separation vector for aircraft of type 1 and 2 is given by $[q(1), q(2)] = [2, 5]$. In Figure 6.2 we see the objective value monotonically decreasing, as we expect. We derive a Monge map from the algorithm output by finding the index of the maximum element of each row. We plot this map (with rectangles representing the minimum separation distance) at the bottom of Figure 6.3. Comparing with the initial distribution at the top of the figure, we see all conflicts have been resolved in the bottom plot, as there are no overlaps in the bars.

Algorithm 7: Row-wise optimisation

Input: List of arrival times, $\boldsymbol{t} \in \mathbb{R}^n$, transport cost matrix C, discretisation scale, N, relevant parameters for the objective function, and a relaxation parameter w, convergence criterion: objective remains within a tolerance of *tol* within the last T steps.

Convert arrival times to a vector in $\{0,1\}^N$

Initialise the transport matrix as the identity, $\mathbb{I}_{N\times N}$

Reduce the dimension of the problem by removing all rows without an aircraft for the transport and the cost matrix to $\mathbb{R}^{n\times N}$, denote the reduced transport matrix $\tilde{\mathbb{I}}$

Find the number of conflicts, $R(\tilde{\mathbb{I}})$ and objective value $\mathcal{L}$ of the starting configuration.

if $R(\tilde{\mathbb{I}}) > 0$ **then**

  while *Last improvement greater than tolerance is within T steps* **do**

    for *Each row* **do**

      Compute the linear term which captures the conflict and transport cost $\boldsymbol{k_i}$. Compute $\boldsymbol{p^*}$ using (6.11).

      Update this row: $\boldsymbol{p_i^{(t+1)}} \leftarrow (1-w)\boldsymbol{p_i^{(t)}} + w\boldsymbol{p^*}$

    end

    Compute and store new objective

    if *All conflicts resolved,* $R(P^{(t+1)}) = 0$ **then**

      break

    end

  end

end

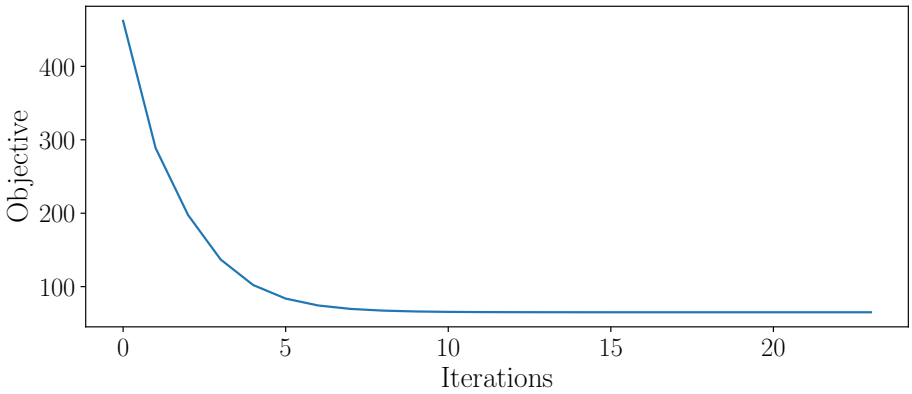


Figure 6.2: An example of the monotonically decreasing objectives for the row-wise algorithm.

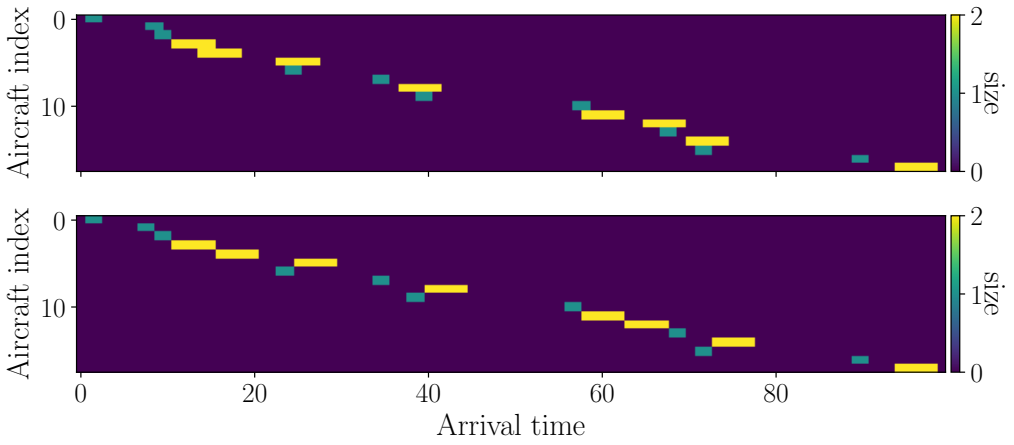


Figure 6.3: An example of application of the row-wise algorithm using a cubic cost with $c = 1$, $\alpha = 40$, $\beta = 1$, $\epsilon = 1$, $w = 0.5$, and the minimum separation vector for aircraft of type 1 and 2 is given by $[q(1), q(2)] = [2, 5]$. The top diagram shows the starting distribution and the bottom diagram shows the converged output. The colour of the bars represent the aircraft type (with 2 representing the larger aircraft type) and the length of the bars represents separation distance. Overlaps in the columns represents conflicts.

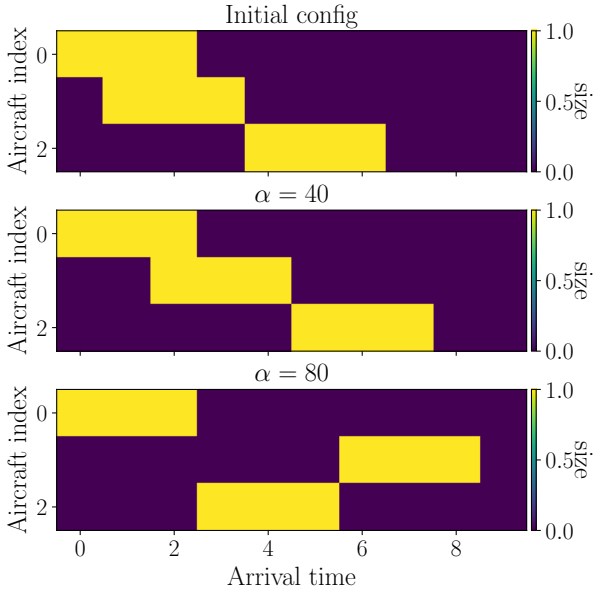
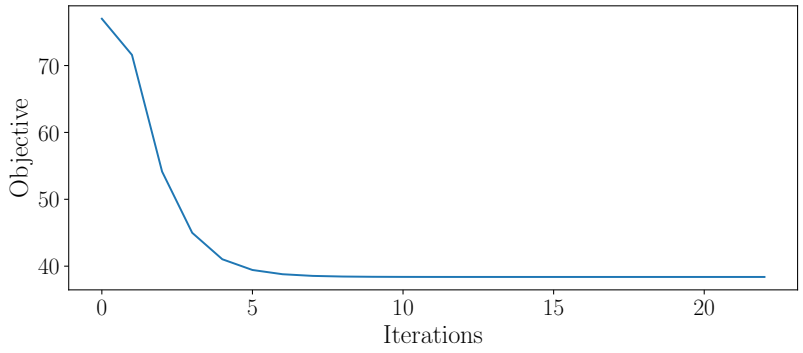
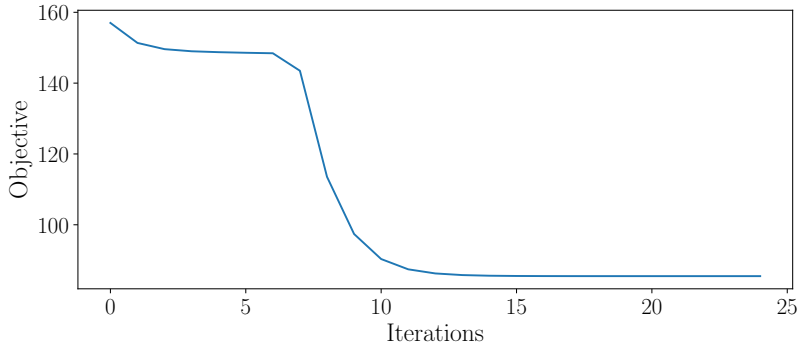


Figure 6.4: A simple three aircraft system. We have cubic cost with $c = 0.8$. Let $\beta = 1$, $\epsilon = 1$ and minimum separation gap of 3. (Top): The initial configuration. (Middle): Result from row-wise algorithm for $\alpha = 40$. (Bottom): Result from row-wise algorithm for $\alpha = 80$. The weight for the updates is set to $w = 0.6$.

However, the problem with this approach is that it is not too difficult to find cases where the algorithm does not work well. Suppose we have three aircraft with an initial distribution as seen in the top of Figure 6.4. All three aircraft are of the same size. Let the transport cost matrix again be the cubic cost with $c = 0.8$. Let $\alpha = 40$, $\beta = 1$, $\epsilon = 1$, and the minimum



(a) Evolution of the objective function for $\alpha = 40$.



(b) Evolution of the objective function for $\alpha = 80$.

Figure 6.5: Objective values for the three plane toy example with two different values for the parameter of the conflict cost α.

separation gap be 3. We see that only the first two aircraft are in conflict with each other. The third one has no overlap with either two. There is no incentive for the third aircraft to move as its initial configuration does not have any conflicts with the other aircraft. Therefore in the middle plot of Figure 6.4 we see that the last aircraft has not moved far enough for all conflicts to be resolved. The reason why we observe a small amount of movement of the last aircraft in the middle diagram is due to the regularisation parameter.

If we increased the penalty coefficient for the number of conflicts up to $\alpha = 80$, then the second aircraft's position becomes increasingly penalised. This causes the second aircraft to move to a slot that is much further to the right, as we observe in the bottom plot of Figure 6.4. This is clearly a sub-optimal solution when much lower costs could be obtained by shifting the third aircraft back a little bit more (i.e. less repositioning of aircraft).

When we look at the objective values in Figure 6.5, we see that when $\alpha = 80$, there is a sharp decrease in the objective when the algorithm starts to deconflict and move further away around the 7th iteration.

To overcome issues like this, we want to incorporate the inconvenience that the last aircraft is causing to the middle aircraft by not moving/changing slots. For this, we introduce some ideas from economics for motivation.

Chapter 7

Economic Interpretation

When optimising each row of our transport matrix separately, we can view each aircraft as a selfish agent who is only optimising their own objective l_i.

The objective function is guaranteed to converge for this row-wise optimisation, since the objective function is monotone decreasing and bounded below. What can we say about the limit point? First order conditions are satisfied for each row, but these are not the first order conditions for the overall objective $L = \sum_i l_i$. This is because the value of each row comes into the objective of the other rows through the column-wise conflict in the value of $\boldsymbol{k_i}$. By optimising on a row-wise basis, these effects have been ignored. As a result, there are no guarantees that the accumulation points of the algorithm are local optima for L.

In the case of the simple three aircraft example in the previous chapter, we see that our point of convergence is certainly not a global optimum for L. In that case, the aircraft which is clogging up the system has no incentive to move, which results in a large increase in the overall objective value as other aircraft have to move around this aircraft, incurring a much larger transport cost than needed.

The arrival management problem can also be thought of as a system of decentralised, autonomous agents (aircraft). By modelling the behaviour of many interacting agents, agent-based models try to characterise the global patterns and behaviour of the system. They have been used in many applications including swarm behaviour [53], modelling the spread of infectious diseases like COVID-19 [90], and in economics [84]. This motivates further exploration of the behaviour of our system on a micro-scale.

For more insight on the row-wise optimisation and potential resolution, we can view our problem from an economic perspective. Row-wise optimisation yields a Nash equilibrium, in the sense that every aircraft has an individually optimized position, given all other aircraft. However, as we see from the simple example of the prisoner's dilemma, a Nash equilibrium may not achieve a social optimum. Each of the m aircraft can be thought of as a consumer of the N slots/goods. The utility function of individuals is given by the row objectives l_i as previously defined (representing the combination of conflict cost, transport cost, and entropy), and $L = \sum_i l_i$ describes a social utility. General equilibrium analysis looks at the ways in which individual decisions may lead to an efficient allocation of goods. A key property to maximise welfare is Pareto optimality.

The case where the behaviour of others can influence the objective of one agent is known as an externality. Externalities are present in our system since agent i cannot directly control the 'consumption' of others, i.e their positioning in time, and vectors p_j, where $j \neq i$ influences l_i. That is, the agent does not have control of where other aircraft move yet their

actions can cause undesirable consequences. Externalities are one of the main reasons in economics for market failure and we will look at the mitigating strategies in this case.

7.1 Background

A feasible economic allocation x is *Pareto efficient/optimal* if there exists no other allocation such that at least one individual has a strictly higher utility and no individual is worse off. Let the utility function of the ith individual be u_i, then Pareto efficiency of x occurs when there are no other allocations $\tilde{x}$ such that $u_i(\tilde{x}) \geq u_i(x)$ for all i and $u_j(\tilde{x}) > u_j(x)$ for some j. We say that an allocation x is locally Pareto optimal if the above is satisfied for any $\tilde{x}$ in a neighbourhood of x.

The canonical example of where maximising selfish/greedy objectives fails to reach a Pareto equilibrium can be seen in the prisoner's dilemma. In this scenario, agents would benefit if there can be enforcement (up to the price of the benefit gained) to help them reach a better outcome. This is the motivation for enforcing public goods/tackling externalities. We return to the prisoner's dilemma later in this section.

7.1.1 Welfare Economics

Definition 7.1.1. *Suppose we are given an allocation x and individual utility functions u_i. The welfare function with weight vector a is given by $W_a(x) = \sum_i a_i u_i(x)$, $a_i > 0$ for all i.*

If we optimise the welfare function $W_a(x)$ then it is easy to see that any maximiser would satisfy Pareto optimality. Negishi [131] also proved that every Pareto efficient solution maximises the welfare function for some a. The set of all Pareto efficient allocations is known as the Pareto frontier.

In an exchange economy, where there is no production of goods, only agents trading with their endowment, suppose there are I agents and J products. Let the allocation be $x = (x_1, x_2, \ldots, x_I)$ where $x_i \in \mathbb{R}^J$ is the consumption vector for the ith agent, and $p \in \mathbb{R}^J$ be a price vector. Let the initial endowment of each agent be $e_i \in \mathbb{R}^J$, $i \in \{1, \ldots, I\}$. A competitive (Walrasian) equilibrium occurs when the following are satisfied.

1. Utility maximisation: for each $i \in \{1, \ldots, I\}$, x_i is a solution to

$$\max_{x_i} u_i(x_i) \quad \text{such that } p \cdot x_i \leq p \cdot e_i.$$

2. Market clearing: for each $l \in \{1, \ldots, J\}$,

$$\sum_i x_{il} = \sum_i e_{il}.$$

As the prices can be scaled linearly, only the relative price matters. We often take the normalised price so that the sum of the prices are 1 (some may refer to this as the price simplex [132]).

The *Invisible Hand* is a concept, usually attributed to Adam Smith [158], which states that in the absence of government, individual optimisation achieves social optimisation. The popular interpretation is that if all producers were free to set their own prices, and consumers free to purchase the goods that they want, from the producers they choose, then prices will settle to a level that benefits both individuals in the system as well as the overall

community. Analogously, the *First Fundamental Theorem of Economics* [14, 59] states that if there are:

1. no externalities;
2. perfect competition;
3. perfect information;
4. rational agents,

then the private market equilibrium is Pareto efficient. Arrow and Debreu showed the existence of such an equilibrium in 1954 [15], provided that the utility functions satisfied conditions such as continuity, monotonicity and concavity.

Market failure describes situations where we do not reach a Pareto efficient equilibrium. One of the most common reasons for market failure is the presence of externalities.

7.1.2 Prisoner's dilemma

As we have alluded above, optimising individual objectives does not necessarily converge to the Pareto equilibrium. Then can be seen clearly in the case of the prisoner's dilemma (Table 7.1).

	cooperate	defect
cooperate	(2,2)	(6,1)
defect	(1,6)	(4,4)

Table 7.1: An example of symmetric prisoner's dilemma

The *Nash equilibrium* for a game is defined as a state where no players can gain any utility by changing their strategy knowing the strategies of all the other players, which are fixed. Let the set of feasible strategies for a single player be denoted by $\mathbb{S}$, and the combined set of strategies of N players be denoted $s = (s_1, s_2, \ldots, s_N)$, where $s_i \in \mathbb{S}$. Let s_{-i} denote the set of strategies without the ith player so that (with a common abuse of notation) $s = (s_i, s_{-i})$. Then, at a Nash equilibrium s^*, we have

$$u(s^*) \geq u(s_i, s^*_{-i}) \quad \forall i \in \{1, \ldots, N\}, \forall s_i \in \mathbb{S}.$$

In a finite game, we have a finite number of players and a finite number of actions to take. In the prisoner's dilemma above, each player can choose (deterministically) to either cooperate or defect. When an action is taken with probability one, this is known as a pure strategy. A mixed strategy describes the cases where the probability is split between actions (though of course the pure strategy can just be considered an edge case of mixed strategies).

Using fixed point theorems [130, 129], Nash proved the existence of at least one mixed Nash equilibrium for every finite game. Wilson's Oddness theorem refined this further [191, 86] by stating that almost all finite games have an odd number of equilibria (that is, a 'typical' finite game will have an odd number of equilibria and this can be proved inductively [191] or geometrically [86]).

In the Prisoner's dilemma, from the perspective of each player, the best/dominant strategy is to defect/betray the other player. As a result the Nash equilibrium corresponds to

the bottom right cell in Table 7.1. However, this choice is not Pareto optimal (in fact, it is the only cell in this table which is not Pareto optimal). Both players would benefit by simultaneously switching to cooperating. In this case, collective group action is needed to get from the Nash equilibrium to the social optimum. The cooperate action can be seen as a social good, since choosing this option will simultaneously benefit all players in the system, but at a lower rate to the each individual compared to acting selfishly by choosing defect. This can also be viewed as an externality, where the action of each agent has unintended consequences on the other agent.

The price of stability (PoS) and the price of anarchy (PoA) are often used to describe the inefficiency of Nash equilibria with respect to the social welfare optimum [108, 136]. The price of stability captures the ratio between the best objective of all Nash equilibria and the social optimum, whilst the price of anarchy captures the ratio of the worst objective among all Nash equilibria and the social optimum. These are designed to be greater than 1, so whether this is defined as Nash equilibrium objective/social optimum objective or social optimum objective/Nash equilibrium objective depends on whether the problem is to minimise cost or maximise welfare. Ideally, the price of stability and the price of anarchy would both be 1, so the deviation from 1 describes the potential gain by having a social planner to influence the behaviour of agents.

There have also been numerous studies in repeated prisoner's dilemma games which resolve the issue by punishment, see e.g. [19, 184]. In [19], a general continuous time, compact action space framework was introduced. Actions were only allowed to be changed countably many times; this subsumes the discrete time/action set-up.

Meanwhile, [184] presents a discrete time study of an iterated Prisoner's dilemma where the players take turns rather than move simultaneously. The degree of cooperation is allowed to take any value in $[0, 1]$, the payoff is also a continuous function of the cooperation level. The goal is to find the optimal (linear) strategy given the actions of the opponent, which is analogous to finding the optimal policy in reinforcement learning. As one might expect, for a fixed static strategy, cooperative strategies lead to the highest payoffs. Strategies were also allowed to adapt based on random search, and Nash equilibria were identified.

7.1.3 Social good and externalities

Motivated by the prisoner's dilemma, we look at ways to facilitate a better collective choice for general continuous problems.

A *public good* [128], G, will increase the utility of everyone, that is $\partial u_i/\partial G > 0 \ \forall i$. However, they typically require investment. Examples of public goods are the national health service and the police force. Individuals of the society will be better off if everyone contributes to the cost of the public good, yet from each individual's perspective, they will be even better off if they alone did not contribute. Often it is not possible or practical to exclude anyone from the benefit of public goods which encourages selfish behaviour on an individual level.

In order to reach a better collective choice, in some settings, a third party (a social planner) optimises a joint function of the agents. Common types of objectives for the social planner are

- Weakest-link [89]: we want to maximise the least well-off individual, that is $W = \min_i u_i$.
- Summation: we want to maximise the overall welfare, that is $W_a = \sum_i a_i u_i$.

Recall from Section 7.1.1 that if the social planner maximises the total welfare, then this results in a Pareto optimum. However if the decision-making is left to the individual then we do not typically achieve a Pareto optimum for the joint objective.

Example 7.1.2. *We describe a simple example where agents can choose to invest in two goods, a private good and a public good. Each individual invests an amount X_i into their private good and G_i in the public good. Let the total invested in the public good be $G = \sum_i G_i$. Let the utility function of agent i be given by $u_i(X_i, G)$. Let the budget constraint of each agent be given by $Y_i = P_X X_i + P_G G_i$.*

The Lagrangian for optimising individual utility is given by

$$u_i(X_i, G) - \lambda(Y_i - P_X X_i - P_G G_i).$$

Then from first order conditions we have

$$\frac{\partial u_i/\partial G}{\partial u_i/\partial X_i} = \frac{P_G}{P_X}. \tag{7.1}$$

This says that the marginal rate of substitution of public for private goods is equal to their price ratio.

If we were to optimise social welfare subject to the joint budget constraint instead, then the Lagrangian is given by

$$\sum_i a_i u_i(X_i, G) - \lambda\left(\sum_i Y_i - P_X X_i - P_G G_i\right)$$

and from the first order conditions we have

$$\sum_i \frac{\partial u_i/\partial G}{\partial u_i/\partial X_i} = \frac{P_G}{P_X}. \tag{7.2}$$

This is the Samuelson condition [160] for Pareto efficient allocation of public goods. To compare with (7.1), we write (7.2) as

$$\frac{\partial u_i/\partial G}{\partial u_i/\partial X_i} = \frac{P_G}{P_X} - \sum_{j \neq i} \frac{\partial u_j/\partial G}{\partial u_i/\partial X_i}. \tag{7.3}$$

Under this problem set-up, $\partial u_i/\partial G > 0$ and $\partial u_i/\partial X_i > 0$, so we see that the marginal rate of substitution of public for private goods for an individual is lower in this case. That is, there is a greater amount invested in the public good when we are optimising the social welfare. Individually rational agents would choose to invest a higher proportion of their income into their own private goods rather than the public good compared with this (Pareto) optimum.

Given a set of identical agents, as the number of players increase, the gap between (7.1) and (7.3) widens. This means that agents optimising their own objective will contribute too little to the community pool. As a result, a system may be set up to enforce a cooperative strategy by issuing suitable punishments, e.g. exclusion from social benefits or increased taxation. There are discussions that greater intervention leads to a decrease in individual intrinsic motivation to act morally [72], though we will not concern ourselves with this here.

An *externality* is defined as an unintended effect from the actions of an individual. This effect could be beneficial or harmful to others. For example, a food factory produces

carbon dioxide as a byproduct, which contributes to climate change and would be a negative externality. An example of a positive externality is the immunisation of individuals leading to a reduced likelihood of community spread of a disease.

Formally, this means that the utility function of an agent depends on variables which an agent cannot control (and are controlled by others in the system). The general formulation in an exchange economy with an allocation vector x is $u_i = u_i(x)$, that is, individual utilities depend on the consumption of others.

The presence of externalities typically means that the competitive equilibrium is no longer Pareto optimal, and as noted earlier, that social welfare will not be maximised when agents are left to their own devices. We look at another simple case to highlight the issue.

Example 7.1.3. *Suppose there are two goods in the system $\{A, B\}$. We have two agents who individually select their own consumption. Agent 1 consumes both products whereas agent 2 only consumes one of those (A say). The consumption of the second product B by agent 1 can still affect the utility of agent 2, which would then be an externality as agent 2 has no control over how much agent 1 consumes B. As a concrete example, products A and B may be milk and cigarettes respectively. Agent 1 is a smoker, whilst agent 2 is not. The use of cigarettes by agent 1 negatively affects agent 2 but as each agent acts independently, agent 2 can only suffer the consequences of this externality.*

Let the consumption of agent i for product j be X_j^i. Then we have $u_1 = u_1(X_A^1, X_B^1)$ and $u_2 = u_2(X_A^2, X_B^1)$. If we were to maximise u_1 subject to some budget constraint then we end up with precisely (7.1) with X_B^1 in the place of G as, from agent 1's perspective, there are only two private goods. To find a Pareto optimal solution of this system (using the welfare function) we form the Lagrangian

$$a_1 u_1(X_A^1, X_B^1) + a_2 u_2(X_A^2, X_B^1) - \lambda\left(\sum_{i\in\{1,2\}, j\in\{A,B\}} (Y_i - P_j X_j^i)\right),$$

for notational convenience set $X_B^2 = 0$. Upon finding the first order conditions and rearranging, we obtain

$$\frac{\partial u_1/\partial X_B^1}{\partial u_1/\partial X_A^1} = \frac{P_B}{P_A} - \frac{\partial u_2/\partial X_B^1}{\partial u_2/\partial X_A^2}. \tag{7.4}$$

This means if B brings about a positive externality ($\partial u_2/\partial X_B^1 > 0$) then agent 1 optimising u_1 is not buying enough of B compared with the Pareto efficient solution, and conversely for a negative externality.

Externalities can be dealt with through the introduction of Pigouvian taxes [144]. In the previous example, if we taxed the consumption of B by

$$-\frac{\partial u_2/\partial X_B^1}{\partial u_2/\partial X_A^2},$$

then we would get back to the Pareto optimal solution. This approach requires knowing the precise model and details of the system.

Another popular concept in the presence of externalities is the Coase theorem [46]. This states that in a system with complete information, assuming there are no transaction costs, we have efficient allocation of resources regardless of the initial property rights. This means that having a property right over the externalities will allow a Walrasian equilibrium to be reached.

In practice, there are often non-negligible transaction costs and so the theorem would be inapplicable. Coasian bargaining has lead to an extensive amount of research as to the scenarios where the theorem applies, see e.g. [76]. Recent work includes [44], which gives necessary and sufficient conditions for the Coase theorem to hold, that is, when do initial property rights not matter. For a polluter-pollutee model, constraints are introduced on the types of utility functions allowed.

A final concept we shall use is *free disposal*, which means that an agent can discard a resource without cost. Alternatively, we can think of this as having a fictional agent in the system who will buy up any free capacity with a price of zero. This means that no agent has to hold anything they do not want.

7.2 Pareto optimality in air traffic control

We derive the conditions for Pareto optimality in our arrival management problem.

7.2.1 Two plane case

In this case the number of conflicts is $R(P) = p_1^\intercal A p_2$ where p_i is a column vector formed by the ith row of P, A is a matrix with the coefficients given as in (6.6) to (6.8). As this is a scalar, we can likewise write $R(P) = p_2^\intercal A^\intercal p_1$.

When we are optimising by row (recall that one row represents one aircraft), e.g. row 1, our optimisation problem is given by

$$\min_{p_1} l_1(p_1, p_2) = \min_{p_1} \left\{ (p_2^\intercal A^\intercal + c_1^\intercal) p_1 + \epsilon p_1^\intercal (\log(p_1) - \mathbb{1}_n) \right\}$$

such that $\mathbb{1}_n^\intercal p_1 = 1$.

When we optimise the social welfare $L = l_1 + l_2$, the first order condition (w.r.t. p_1) becomes

$$A p_2 \ + c_1 + \epsilon \log(p_1) + A p_2 = 0.$$

Therefore the only difference between optimising the social welfare L and optimising each agent is the addition of the last term, which has the effect of doubling the weight on the conflict cost. We observed before that the conflict cost introduces a non-convexity in our optimisation problem, so (as seen in Figure 6.4) this may not lead to improvements in our results. From the externalities example in the previous section, we can introduce a Pigouvian tax on player 1, penalising the 'consumption' of the first row by an amount Ap_2 to achieve a Pareto optimum. However, this will not generally lead to a social optimum.

7.2.2 General case

For the general case we can consider the total number of conflicts as the conflicts between pairs of aircraft and apply the above. That is,

$$R(P) = \sum_{i<j} p_j^\intercal A_{ij} p_i$$

so then

$$l_i = \left(\sum_{i \neq j} p_j^\intercal A_{ij} + c_i^\intercal \right) p_i + \epsilon p_i^\intercal (\log(p_i) - \mathbb{1}_n).$$

By optimising the sum of l_i, for the first order condition (w.r.t. p_1), we end up with

$$\sum_{i \neq j} A_{ji} p_j + c_i + \epsilon \log(p_1) + \sum_{i \neq j} A_{ji} p_j = 0,$$

where $A_{ji} = A_{ij}^{\intercal}$.

Note that there are no structural changes between the two-player and the general case; there are again additional terms to double the penalty of conflicts. Given that we choose constants to indicate the importance of conflict vs transport costs, this means that row-wise optimisation is a coordinate descent-like method for a problem with half the weight on the conflicts, and therefore it is Pareto optimal, for this modified problem.

Chapter 8

Multi-agent games

8.1 A simple 2 player, 2 action game

Consider the classic simple case of having two agents/players, agent A and agent B, and only two actions to choose from, action 1 and action 2. Agents are trying to choose the action which would minimise their own costs. The payoff bi-matrix can be seen in Table 8.1. Depending on the relative values of these entries, we get different behaviours for the agents. We can compute equilibrium points, Pareto optimal points, Nash equilibria and so on. We begin by looking at the case of simultaneous movement of non-communicating agents in this section.

		Agent B	
		1	2
Agent A	1	(a_{11}, b_{11})	(a_{12}, b_{12})
	2	(a_{21}, b_{21})	(a_{22}, b_{22})

Table 8.1: Payoffs for a general two-player two-action game.

In particular, in the context of optimising aircraft landings, we consider a case with conflict cost of k, which means that for sufficiently high k, players should avoid choosing the same actions. That is, agents want to minimise their costs and should choose an anti-coordination strategy. Suppose that each agent also has a preference for one of the actions irrespective of the other player. For the anti-coordination setting we should have either $a_1 < a_2$ and $b_1 < b_2$, or $a_1 > a_2$ and $b_1 > b_2$ (that is, the agents prefer the same action). So let us assume without loss of generality that $a_1 < a_2$ and $b_1 < b_2$, so that both agents prefer action 1. The payoffs are given in Table 8.2.

		Agent B	
		1	2
Agent A	1	$(a_1 + k, b_1 + k)$	(a_1, b_2)
	2	(a_2, b_1)	$(a_2 + k, b_2 + k)$

Table 8.2: Costs of actions in a two player anti-coordination game.

Our aim is to develop a method such that, even when individual agents optimise only

their own objective function, this still converges to the social optimum, which we define to be the pure strategy with the minimum sum of costs. This is an engineered problem where we are free to choose the conflict costs k. We assume that $k \geq 0$.

If there are no conflict costs (that is $k = 0$), then two independent players that have their own action preferences would simply choose the action leading to the lowest cost for them, and this naturally finds the social optimum. When there are no conflict costs, the problem is convex.

Now suppose that the problem changes and the players are no longer independent. Their outcome depends on the action of the other player so that there is a conflict cost ($k > 0$). The introduction of conflict costs breaks the convexity, and it is no longer guaranteed that selfish agents will end up at the ideal social outcome when acting independently. The problem of how to optimise their mixed strategy is of interest.

Looking at the mixed strategy representation, let us denote the probability of choosing action 1 for agent A and agent B be p and q respectively. Suppose that the agents try to improve upon their current strategies through marginal changes, modelled by gradient flow in strategy space.

Let the cost function of each individual agent be l_A and l_B, these are given by

$$\begin{aligned} l_A &= pq(a_1 + k) + p(1-q)a_1 + (1-p)qa_2 + (1-p)(1-q)(a_2 + k), \\ l_B &= pq(b_1 + k) + p(1-q)b_2 + (1-p)qb_1 + (1-p)(1-q)(b_2 + k), \end{aligned}$$

and partial derivatives given by

$$\frac{\partial l_A}{\partial p} = q(a_1 + k) + (1-q)a_1 - qa_2 - (1-q)(a_2 + k) = (2q-1)k - (a_2 - a_1), \tag{8.1}$$

$$\frac{\partial l_B}{\partial q} = p(b_1 + k) + (1-p)b_1 - pb_2 - (1-p)(b_2 + k) = (2p-1)k - (b_2 - b_1). \tag{8.2}$$

Note that $a_2 - a_1$ represents the gain for agent A by choosing action 1 over action 2. The sign of other term in $\frac{\partial l_A}{\partial p}$, $(2q-1)k$, depends on whether the belief that the other agent will choose action 1 is greater than $1/2$ or not. If they are more likely to choose action 1, then the derivative $\frac{\partial l_A}{\partial p}$ is positive for agent A with increasing k (that is, agent A incurs a greater cost in that direction).

As we consider each agent moving independently, we can write this as an autonomous system

$$\begin{pmatrix} \dot{p} \\ \dot{q} \end{pmatrix} = \begin{pmatrix} 0 & 2k \\ 2k & 0 \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} - \begin{pmatrix} k + a_2 - a_1 \\ k + b_2 - b_1 \end{pmatrix}. \tag{8.3}$$

By rearranging, we find that the stationary point is given by

$$\left(\frac{k + b_2 - b_1}{2k}, \frac{k + a_2 - a_1}{2k} \right). \tag{8.4}$$

Note that as the problem lives on a unit square, there may not be an interior equilibrium point within this constrained space, if $b_2 - b_1 > k$ or $a_2 - a_1 > k$. Note that the flow changes on the boundary as our updated states are clipped between 0 and 1 for p and q to ensure the strategies remain on the unit square (i.e. a projection back onto the unit square.)

To find the eigenvalues, we first note a general result.

Example 8.1.1. *The eigenvalues of*

$$A = \begin{pmatrix} 0 & x \\ x & 0 \end{pmatrix} \tag{8.5}$$

are x and $-x$. The eigenvalues can be obtained by inspection, as the trace and the determinant gives the sum and product of the eigenvalues respectively. By solving the equation $Av = \lambda v$ for $\lambda = x$ and $\lambda = -x$, we find that the eigenvectors are $\binom{1}{1}$ and $\binom{1}{-1}$ respectively.

Therefore, in our system (8.3), the eigenvalues are $2k$ and $-2k$, with eigenvectors $\binom{1}{1}$ and $\binom{1}{-1}$ respectively.

Rather than each agent optimising individually, we can consider a planner who optimises (using a gradient method) the welfare function given by

$$\begin{aligned} L &= l_A + l_B \\ &= pq(a_1 + b_1 + 2k) + p(1-q)(a_1 + b_2) + (1-p)q(a_2 + b_1) + (1-p)(1-q)(a_2 + b_2 + 2k), \end{aligned}$$

with partial derivatives

$$\begin{aligned} \frac{\partial L}{\partial p} &= q(a_1 + 2k) + (1-q)a_1 - qa_2 + -(1-q)(a_2 + 2k) = 2(2q-1)k + a_1 - a_2, \\ \frac{\partial L}{\partial q} &= p(b_1 + 2k) - pb_2 + (1-p)b_1 - (1-p)(b_2 + 2k) = 2(2p-1)k + b_1 - b_2. \end{aligned}$$

We see that in this case, we have just doubled the coefficient on the terms involving k and the result is qualitatively the same as considering independent agents. Again this can be written as an autonomous system:

$$\begin{pmatrix} \dot{p} \\ \dot{q} \end{pmatrix} = \begin{pmatrix} 0 & 4k \\ 4k & 0 \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} - \begin{pmatrix} 2k + a_2 - a_1 \\ 2k + b_2 - b_1 \end{pmatrix},$$

where the stationary point is given by

$$\left(\frac{2k + b_2 - b_1}{4k}, \frac{2k + a_2 - a_1}{4k} \right). \tag{8.6}$$

Note that this is precisely the Nash equilibrium (and we can similarly derive this by equating the outcome of choosing action 1 and action 2 for each agent).

Similar to the case when the agents are optimising their own objectives, the equilibrium point will not be on the unit square (within the domain of interest), if $b_2 - b_1 > 2k$ or $a_2 - a_1 > 2k$. Also we can see from this autonomous representation that, by Example 8.1.1, although the eigenvalues are now $4k$ and $-4k$, the eigenvectors remain unchanged. As the inherent structure of the problem has not changed, simply optimising the social cost directly using gradient flow is not sufficient as a solution to find the social optimum.

Recall that we suppose $a_1 < a_2$ and $b_1 < b_2$, so that both agents prefer action 1. Let us further suppose that $a_1 + b_2 > a_2 + b_1$, that is $b_2 - b_1 > a_2 - a_1$, which means agent B has more to gain by choosing action 1 compared with agent A. Therefore the social optimum would be $(p, q) = (0, 1)$.

8.1.1 Considerate cost formulation

Now suppose that we add some 'considerate' cost term to help the flow formed through gradient descent steps of the individual agents converge to the social optimum, as motivated by the individual Pigouvian tax mentioned previously. We add costs as seen in Table 8.3, where $\gamma > 0$ is an additional parameter that we are free to choose. For example, the social cost associated with agent A choosing action 1 is γs_1.

		Agent B	
		1	2
Agent A	1	$(k + a_1 + \gamma s_1, k + b_1 + \gamma t_1)$	$(a_1 + \gamma s_1, b_2 + \gamma t_2)$
	2	$(b + \gamma s_2, b_1 + \gamma t_1)$	$(k + a_2 + \gamma s_2, k + b_2 + \gamma t_2)$

Table 8.3: Two-player example with the addition of a considerate cost term.

In the context of anti-coordination games, agent A and agent B should avoid choosing the same action to avoid the conflict cost. So if agent A wants to increase p and make choosing action 1 more likely, then this would be an externality on agent B, as their 'conflict' penalty would now increase if agent B also wants to choose action 1. Whether agent 2 (locally) prefers action 1 or action 2 is given by their partial derivatives. Since we only have two actions, these partial derivatives will be equal in magnitude and have opposite signs. So agent B prefers the action with the negative partial derivative.

This additional cost does not need to encourage agents to take actions that no-one else wants (in a resource allocation setting with a free disposal assumption, a fictional agent will buy these resources up), so our costs should be positive. Hence we choose

$$\begin{aligned} s_1 &= \max\left(\frac{\partial l_B}{\partial q}, 0\right), \\ s_2 &= \max\left(-\frac{\partial l_B}{\partial q}, 0\right), \\ t_1 &= \max\left(\frac{\partial l_A}{\partial p}, 0\right), \\ t_2 &= \max\left(-\frac{\partial l_A}{\partial p}, 0\right). \end{aligned}$$

Note that in this case, due to having opposite signs inside the non-linear function (taking the maximum), one of each pair of considerate costs will be zero. This means that we are adding a penalty onto the action that is favoured by the other agent (and the nonlinear term vanishes). Using (8.1) and (8.2), we simplify the new derivative function to

$$\begin{aligned} \frac{\partial \tilde{l}_A}{\partial p} &= \frac{\partial l_A}{\partial p} - \gamma \frac{\partial l_B}{\partial q} \\ &= -2\gamma kp + 2kq + a_1 - a_2 - \gamma(b_1 - b_2) + (\gamma - 1)k, \\ \frac{\partial \tilde{l}_B}{\partial p} &= \frac{\partial l_B}{\partial q} - \gamma \frac{\partial l_A}{\partial p} \\ &= 2kp - 2\gamma kq - \gamma(a_1 - a_2) + b_1 - b_2 + (\gamma - 1)k. \end{aligned}$$

Therefore when the agents are optimising on an individual basis with this considerate cost, we obtain the following autonomous system

$$\begin{pmatrix}\dot{p}\\ \dot{q}\end{pmatrix} = \begin{pmatrix}-2\gamma k & 2k\\ 2k & -2\gamma k\end{pmatrix}\begin{pmatrix}p\\ q\end{pmatrix} + k\begin{pmatrix}\gamma-1\\ \gamma-1\end{pmatrix} + \begin{pmatrix}1 & -\gamma\\ -\gamma & 1\end{pmatrix}\begin{pmatrix}a_1-a_2\\ b_1-b_2\end{pmatrix}. \tag{8.7}$$

Solving the above system for $\binom{\dot{p}}{\dot{q}} = \binom{0}{0}$, we find that the equilibrium point has not changed from $(p,q) = \left(\frac{k+b_2-b_1}{2k}, \frac{k+a_2-a_1}{2k}\right)$, that we obtained by optimising on an individual basis. We can think of this as improving the 'monotonicity' of the flow (in the sense of [32], as if $x = (p,q)^\intercal$ then for γ large, $\langle x-\tilde{x}, \dot{x}-\dot{\tilde{x}}\rangle < 0$) so we get a convergent acyclic system.

To check the eigenvalues and eigenvectors, we again look at a more general example.

Example 8.1.2. *Previously, we noted that, for the matrix*

$$\begin{pmatrix}0 & x\\ x & 0\end{pmatrix},$$

the eigenvalues are x and $-x$, with eigenvectors $\binom{1}{1}$ and $\binom{1}{-1}$ respectively. If the matrix is changed to be

$$\begin{pmatrix}y & x\\ x & y\end{pmatrix}, \tag{8.8}$$

then the eigenvalues become $y+x$ and $y-x$, but the eigenvectors remain $\binom{1}{1}$ and $\binom{1}{-1}$ respectively.

Therefore with the considerate costs, the eigenvalues are now given by $\lambda = -2k(\gamma \mp 1)$ for $\gamma \neq 1$, but the eigenvectors are unchanged. Looking at the eigenvalues, we note that the stationary point changes to an unstable node when $\gamma > 1$. At $\gamma = 1$, we have a zero eigenvalue and hence a line of stationary points (the nullspace is non-trivial).

To understand the behaviour of this system, it is useful to identify its isoclines. Isoclines are subspaces of the phase space where the derivative of one of the variable with respect to time is not changing. Nullclines is the more specific example where this derivative is zero. An equilibrium occurs when all nullclines intersect.

The nullclines of our system with considerate costs are given by

$$q = \gamma p + \frac{1}{2k}\big((a_2-a_1) + \gamma(b_1-b_2) + k(\gamma-1)\big), \tag{8.9}$$

$$q = \frac{1}{\gamma}p + \frac{1}{2\gamma k}\big(-\gamma(a_1-a_2) + (b_1-b_2) + k(\gamma-1)\big). \tag{8.10}$$

Under our assumptions, the social optimum is $(p,q) = (0,1)$. We derive the conditions to impose on k and γ to ensure that this is always achieved.

Proposition 8.1.3. *Suppose that $k < b_2 - b_1$ and either*

$$\frac{a_2-a_1-k}{b_2-b_1-k} < \gamma < \min\left(\frac{b_2-b_1-k}{a_2-a_1+k}, \frac{a_2-a_1+k}{b_2-b_1-k}\right),$$

or

$$\max\left(\frac{b_2-b_1-k}{a_2-a_1+k}, \frac{a_2-a_1+k}{b_2-b_1-k}\right) < \gamma < \frac{b_2-b_1+k}{a_2-a_1+k},$$

alternatively, suppose

$$k < \frac{(b_2 - b_1) - (a_2 - a_1)}{2},$$

and

$$\frac{(a_2 - a_1) + k}{(b_2 - b_1) - k} < \gamma < \frac{(b_2 - b_1) - k}{(a_2 - a_1) + k},$$

then there is no basin of attraction of the $(p, q) = (1, 0)$ *corner, and all trajectories of the considerate flow converge to* $(p, q) = (0, 1)$.

The proof of this proposition can be found in Appendix G, where more specific ranges on k are also stated to ensure that there are feasible values for γ.

A qualitative explanation as to why the above ranges ensure the correct convergence, is that by changing k and γ, we are making changes to the vector field. From (8.6) we see that the location of the equilibrium point of the system is dependent on k. As $k \to \infty$, the equilibrium point (an unstable node for gradient descent) tends to (0.5,0.5). That is, suppose it is prohibitively expensive to collide, then with no information about the other agent, having this mixed strategy is an equilibrium by symmetry. However, given any slight perturbation, i.e. belief/information about what the other agent might do, the strategy of the agent would immediately change (hence this is unstable). For smaller values of k, a perturbation in k has the effect of 'translating' the vector field, and we are only interested in what happens with the vector field within the $[0, 1] \times [0, 1]$ space.

The gradient of the nullclines in (8.9) and (8.10) depends on γ. Therefore as we vary γ, the gradient of the nullclines changes and we can effectively manipulate the direction of the vector field at any location, see Figure 8.1. In Figure 8.1, only the top left diagram satisfies the conditions in Proposition 8.1.3, and we can see this is indeed the only one where the direction of the vector field ensures trajectories can only be absorbed at the $(p, q) = (0, 1)$ strategy.

Note that the eigenvectors are unaffected by changes in k and γ, therefore the state space is still separated into sections by the eigenvectors from the equilibrium point, as trajectories typically cannot cross these. The only exception to this occurs at the boundaries, where a projection back onto the unit square allows the trajectory to move through eigenvectors. That is, when our updated states are clipped between 0 and 1 for p and q. The conditions that we have derived in Proposition 8.1.3 is such that the vector field at the boundaries would ensure convergence to the socially optimum pure strategy. This is the intuition behind the proof in Appendix G. Hence, within the ranges specified in Proposition 8.1.3, the vector fields point away from the sub-optimal anti-coordination strategy and gradient-based optimisation converges to the 'correct' socially optimal one.

Finally we note that in the above analysis, we supposed that $a_1 + b_2 > a_2 + b_1$, therefore agent B has more to gain by choosing action 1 rather than 2. By ensuring the above conditions, individuals following their own gradient descent will converge to the social optimum $(p, q) = (0, 1)$. What if $a_1 + b_2 < a_2 + b_1$ so that agent A has more to gain? We note that this case is symmetric to the case analysed above, the roles of agent A and agent B are simply interchanged.

Remark 8.1.4. *Recall that k in this two-player example is analogous to the conflict cost in the air traffic control problem. This result shows that having a large k is not necessarily a good idea, even if we wanted to prohibit these conflicts.*

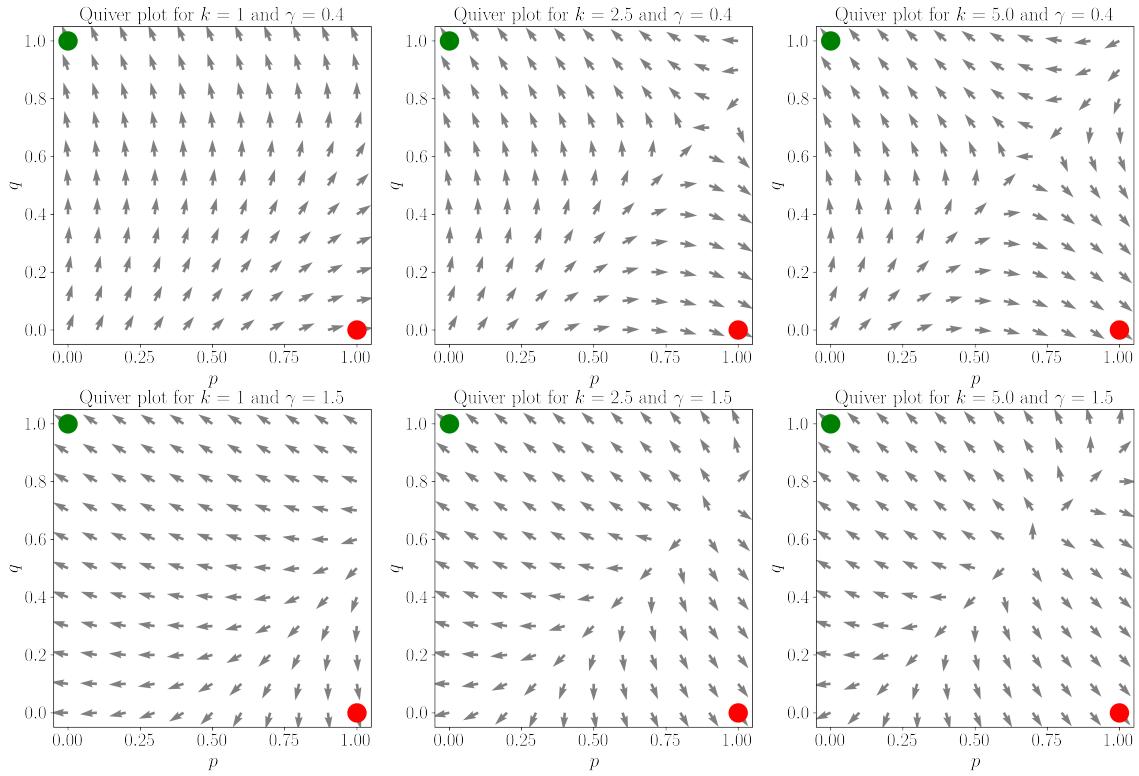


Figure 8.1: Effect of changing k and γ on the vector field, note that the arrows are not scaled by magnitude so that the direction is clearer. For these plots, only the top left one satisfies the conditions from Proposition 8.1.3 and therefore all trajectories converge to $(0, 1)$ as desired.

8.2 Generalising the 2 player case to M actions

Next we extend our analysis to two-player games with M actions. Let the probability that action j is chosen by agent A and agent B be given by p_j and q_j respectively. Let the vector of probabilities for each agent $\boldsymbol{p}$ and $\boldsymbol{q}$ be in $\mathcal{S}_M$, the simplex in $\mathbb{R}^M$ (5.1).

For greater numbers of actions, it quickly becomes infeasible to perform an analysis as seen in Appendix G. Nevertheless, we can reduce our overall problem into many two-action subproblems. We want to find the social optimum, by ensuring convergence on each subgame, and finding suitable conditions and sampling strategies to choose these subgames. For each player there are $\binom{M}{2}$ pairs of actions, and so $\binom{M}{2}^2$ possible subgames.

Assume for algebraic convenience that there are only conflict costs if the agents choose the same actions. For simplicity let us suppose the actions are given in preference order for each agent so that $a_1 \leq a_2 \leq \cdots \leq a_M$ and $b_1 \leq b_2 \leq \cdots \leq b_M$.

Then the subgames reduce to a few cases. The first case is where the actions chosen are the same, that is, there are conflict costs along the diagonal of the bi-matrix of the subgame, this can be seen in Figure 8.2. This is precisely the case we have analysed for the two-action case in Section 8.1.

There is also the case where there are no overlaps at all between the chosen actions and hence no conflict penalty. In this case the players are free to choose the action that will be cheapest to them. The problem is convex and a simple gradient method will reach the

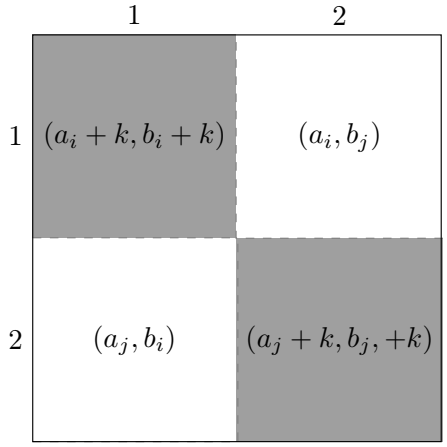


Figure 8.2: Reduced subproblem where the two actions chosen are the same.

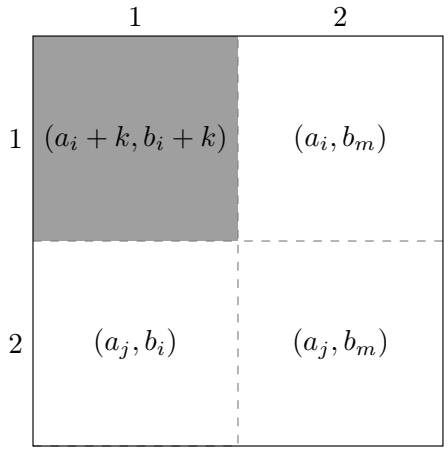


Figure 8.3: Reduced subproblem where the preferred action of the two agent is the same. Note that $i < j$ and $i < m$, therefore by assumption $a_i < a_j$ and $b_i < b_m$.

desired solution.

When just one of the actions overlaps, the behaviour depends on which action this is. If it is not the action with the minimum cost for both players, then we still have a convex problem and again a simple gradient method will suffice. However if this overlapping action is the preferred action for both agents (as seen in Figure 8.3, recall that $a_i < a_j$, $b_i < b_j$ for $i < j$), then again we need to impose conditions to ensure convergence to the social optimum. We can perform a similar analysis to Section 8.1 and find the equivalent conditions.

Let us suppose as before that $b_m - b_i > a_j - a_i$, which means that agent B has more to gain by choosing action i than agent A. Otherwise we can interchange the players since the problem is symmetric.

We have

$$\begin{aligned} l_A &= pq(a_i + k) + p(1-q)a_i + (1-p)qa_j + (1-p)(1-q)a_j \\ &= pa_i + pqk + (1-p)a_j, \\ l_B &= qb_i + pqk + (1-q)b_m, \\ L &= pa_i + qb_i + (1-p)a_j + (1-q)b_m + 2pqk, \end{aligned}$$

and

$$\begin{aligned} \frac{\partial l_A}{\partial p} &= a_i - a_j + qk, \\ \frac{\partial l_B}{\partial q} &= b_i - b_m + pk, \\ \frac{\partial L}{\partial p} &= a_i - a_j + 2qk, \\ \frac{\partial L}{\partial q} &= b_i - b_m + 2pk. \end{aligned}$$

Written as an autonomous system, we obtain,

$$\begin{pmatrix} \dot{p} \\ \dot{q} \end{pmatrix} = \begin{pmatrix} 0 & k \\ k & 0 \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} + \begin{pmatrix} a_i - a_j \\ b_i - b_m \end{pmatrix}.$$

The equilibrium point is $\left(\frac{b_m - b_i}{k}, \frac{a_j - a_i}{k}\right)$. This is again in the format of Example 8.1.1 so the eigenvalues are k and $-k$ with eigenvectors $\binom{1}{1}$ and $\binom{1}{-1}$ respectively.

We impose the considerate cost only for actions that would lead to conflicts, similar to our approach in Section 8.1.

Let

$$\begin{aligned} \frac{\partial \tilde{l}_A}{\partial p} &= \frac{\partial l_A}{\partial p} - \gamma \frac{\partial l_B}{\partial q} = a_i - a_j + qk - \gamma(b_i - b_m + pk), \\ \frac{\partial \tilde{l}_B}{\partial q} &= \frac{\partial l_B}{\partial q} - \gamma \frac{\partial l_A}{\partial p} = b_i - b_m + pk - \gamma(a_i - a_j - qk), \end{aligned}$$

so we have

$$\begin{aligned} \begin{pmatrix} \dot{p} \\ \dot{q} \end{pmatrix} &= \begin{pmatrix} -\gamma k & k \\ k & -\gamma k \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} + \begin{pmatrix} a_i - a_j \\ b_i - b_j \end{pmatrix} + \gamma \begin{pmatrix} b_i - b_j \\ a_i - a_j \end{pmatrix} \\ &= \begin{pmatrix} -\gamma k & k \\ k & -\gamma k \end{pmatrix} \begin{pmatrix} p \\ q \end{pmatrix} + \begin{pmatrix} 1 & -\gamma \\ -\gamma & 1 \end{pmatrix} \begin{pmatrix} a_i - a_j \\ b_i - b_m \end{pmatrix}. \end{aligned}$$

By Example 8.1.2, we see that the eigenvalues become $-\gamma k \pm k$, with eigenvectors remaining $\binom{1}{1}$ and $\binom{1}{-1}$ respectively.

Next we derive the equilibrium:

$$\begin{aligned} \begin{pmatrix} p^* \\ q^* \end{pmatrix} &= \begin{pmatrix} -\gamma k & k \\ k & -\gamma k \end{pmatrix}^{-1} \begin{pmatrix} 1 & -\gamma \\ -\gamma & 1 \end{pmatrix} \begin{pmatrix} a_j - a_i \\ b_m - b_i \end{pmatrix} \\ &= \frac{1}{k^2(\gamma^2 - 1)} \begin{pmatrix} -\gamma k & -k \\ -k & -\gamma k \end{pmatrix} \begin{pmatrix} 1 & -\gamma \\ -\gamma & 1 \end{pmatrix} \begin{pmatrix} a_j - a_i \\ b_m - b_i \end{pmatrix} \\ &= \frac{1}{k^2(\gamma^2 - 1)} \begin{pmatrix} 0 & k(\gamma^2 - 1) \\ k(\gamma^2 - 1) & 0 \end{pmatrix} \begin{pmatrix} a_j - a_i \\ b_m - b_i \end{pmatrix} \\ &= \frac{1}{k} \begin{pmatrix} b_m - b_j \\ a_j - a_i \end{pmatrix}, \end{aligned}$$

so again the equilibrium is unchanged by the addition of this social cost.

We find the conditions for the desired convergence.

Proposition 8.2.1. *Suppose that $k < b_m - b_i$ and either*

$$\frac{a_j - a_i - k}{b_m - b_i - k} < \gamma < \min\left(\frac{b_m - b_i - k}{a_j - a_i}, \frac{a_j - a_i}{b_m - b_i - k}\right)$$

or

$$\max\left(\frac{b_m - b_i - k}{a_j - a_i}, \frac{a_j - a_i}{b_m - b_i - k}\right) < \gamma < \frac{b_m - b_i}{a_j - a_i},$$

alternatively

$$\frac{a_j - a_i}{(b_m - b_i) - k} < \gamma < \frac{(b_m - b_i) - k}{a_j - a_i}$$

and

$$0 < k < (b_m - b_i) - (a_j - a_i),$$

then there is no basin of attraction of the $(p, q) = (1, 0)$ corner, and all trajectories of the considerate flow (Algorithm 8) converge to $(p, q) = (0, 1)$.

The proof of this proposition can be found in Appendix H, with further specifications on k to ensure that there are feasible values of γ to satisfy the inequalities.

Algorithm 8: Algorithm for the M action case

Input: Initial distribution of probability for each agent p, $q \in \mathcal{S}_M$.

for *each iteration up to the maximum number of iterations* **do**

- Randomly select 2 actions from $\{1, 2, ..., M\}$ for each agent either uniformly or based on the vectors p and q.
- Find the probability of those actions $\hat{p} = [p_i, p_j]$, $\hat{q} = [q_l, q_m]$.
- Define a 2-action subgame by rescaling the probability of the two actions so that the vectors $\hat{p}, \hat{q}$, are in $\mathcal{S}_2$.
- **if** *both actions are the same* **then**
  - Check if k and γ satisfies the conditions in Proposition 8.1.3 and enforce the constraints if required.
- **else if** *there is an overlap of the smaller action index* **then**
  - Check if k and γ satisfies the conditions in Proposition 8.2.1 and enforce the constraints if required.
- **else**
  - The problem is convex, no constraints are needed.
- Run individual gradient descent until convergence to a pure strategy (projecting back to the unit square if needed).
- Rescale the probability back to the original scale.
- Update the multi-action probability by taking a gradient step, where the step size is limited by $\min(p_i + p_j, q_l + q_m)$.

end

Now that we have obtained conditions to ensure convergence to the social optimum for any possible subgames, we formulate the method seen in Algorithm 8. In Algorithm 8, we limit the step size of the updates made by agents to the smallest step that either one of them can take. This is to ensure stability.

Let us think about the 3-action case to gain some intuition. When we choose a pair of actions and wish to change the probability of choosing those 2 actions, we are keeping the probability of the third action fixed. We can think of this as each agent is being updated in their separate 2-simplex.

Lemma 8.2.2. *For a 3-action example using Algorithm 8, let* $\epsilon = \min(p_i + p_j, q_l + q_m)$. *At each step, the distance from a pure strategy increases by no more than* $(1 - \sqrt{3}/2)\sqrt{2}\epsilon$.

Proof. When we iterate, we keep the value in one of the dimensions constant. Suppose that the coordinate in the simplex is (x, y, z), where $x + y + z = 1$. Let z be the dimension that we keep constant and let $\epsilon = 1 - z$. When we update the probability of taking 2 actions, we are moving along the solid line shown in Figure 8.4.

By simple trigonometry, the maximum distance from the pure strategy corner $(0, 0, 1)$ is $\sqrt{2}\epsilon$, and the minimum distance is $\sqrt{6}/2\epsilon$ as illustrated in Figure 8.5.

Therefore, in any update, the distance increases at most from $\sqrt{6}/2\epsilon$ to $\sqrt{2}\epsilon$, i.e. a maximum increase of $(1 - \sqrt{3}/2)\sqrt{2}\epsilon$. ■

So we see that even if one dimension is fixed, we may still increase the distance away from a pure strategy at each iteration. In practice this resolves the instability issue caused by taking random pairs of actions at each step of this algorithm.

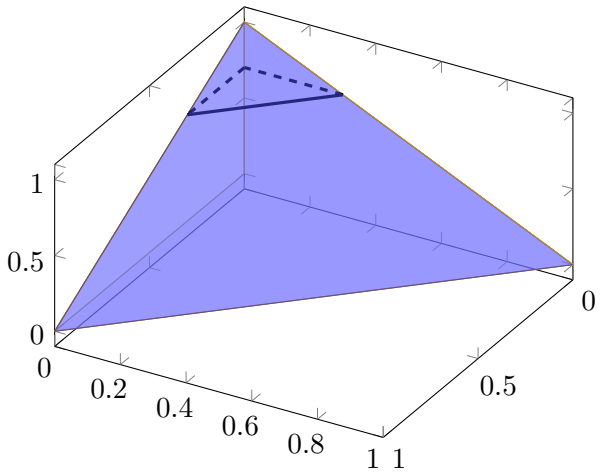


Figure 8.4: Depiction of the intersection between the 2-Simplex and some plane $z = 1 - \epsilon$.

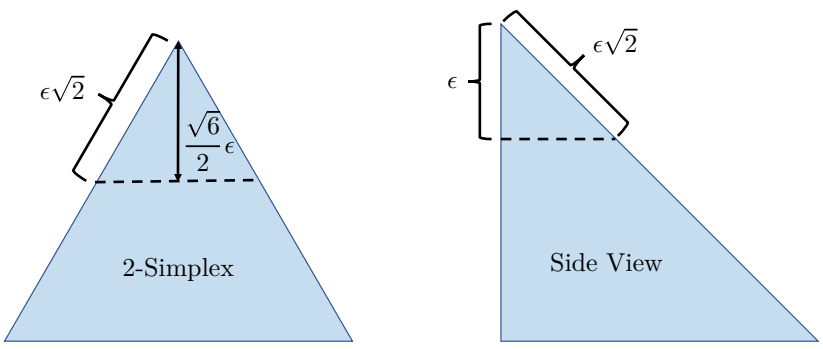


Figure 8.5: View of the 2-simplex and side view of the tetrahedron.

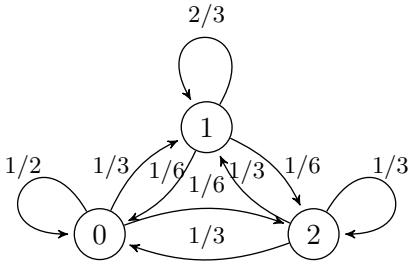


Figure 8.6: Agent A's Markov chain representation.

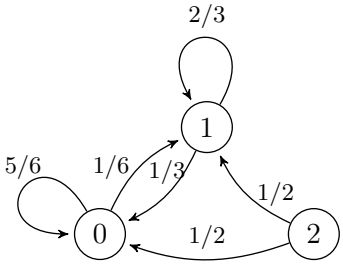


Figure 8.7: Agent B's Markov chain representation.

We now show why this limit is needed. Without this limit, the states of the two agents can be thought of as two Markov chains. Since actions are selected on a random basis, the distribution of the agents will tend towards an equilibrium distribution (as there are finite number of states, so there will be an irreducible set of states with a unique stationary distribution). This equilibrium distribution will depend on the 'exploratory' policy (which is uniform in this case).

Let us illustrate what we mean by this with a simple example. Suppose that we have three actions, and the costs of these three actions (actions 0, 1, and 2) are $[0, 12, 21]$ and $[14, 21, 42]$ for agents A and B respectively. There are $\binom{3}{2}^2 = 9$ possible subgames.

The proportion of transitions for all 9 subgames can be represented as a Markov chain for each agent, we obtain Figures 8.6 and 8.7. The Markov chain for agent A will then have stationary distribution $[3/10, 5/10, 2/10]$ and for agent B, the stationary distribution is $[2/3, 1/3, 0]$. Therefore uniformly sampling the actions for each agent, without limiting the step size, would not converge to the social optimum ($[1, 0, 0]$ and $[0, 1, 0]$ here).

8.3 Generalising to N agents

Finally we consider the case of when there are N agents and M actions, where typically $M > N$.

Again, as it is infeasible to algebraically analyse this case, we simplify these to simple 2-player, 2-action games. At each stage, we randomly pick an agent and sample 2 actions based on its current strategy. We then choose the other agent as the one who is most likely to make the 2 actions selected. The details can be seen in Algorithm 9.

We can also make a modification to Algorithm 9 and show that finding the social optimum is robust to some randomness. Instead of deterministically picking the agent which has the greatest probability of taking the 2 chosen actions, we can sample for the other

agent based on the sum of their probability. That is, we compute $[p_i^1 + p_j^1, \ldots, p_i^N + p_j^N]$ and normalise this vector, then we use this normalised vector to sample for the second agent instead.

Algorithm 9: Algorithm for the N agent case

Input: Initial distribution of probability for each agent $p^1, p^2, \ldots, p^N \in \mathcal{S}_M$, number of max iterations, threshold.

for *each iteration up to the maximum number of iterations* **do**

- Randomly select 1 agent, which we call agent r_1.
- Randomly select 2 actions from $\{1, 2, ..., M\}$ for this chosen agent based on its current distribution over actions, and find the probability of those actions $\hat{p}^{r_1} = [p_i^{r_1}, p_j^{r_1}]$.
- Choose the other agent, agent r_2, by finding $\text{argmax}_r p_i^r + p_j^r$ or by sampling from an empirical distribution based on the vector $p_i^r + p_j^r$, and obtain $\hat{p}^{r_2} = [p_i^{r_2}, p_j^{r_2}]$.
- **if** $p_i^{r_2} + p_j^{r_2} <$ *threshold* **then**
  - Allow first selected agent to optimise as pleased as the other agent does not have enough 'say'.
- **else**
  - Define a 2-action subgame by rescaling the probability of the two actions so that the vectors $\hat{p}^{r_1}, \hat{p}^{r_2}$, are in $\mathcal{S}_2$. Solve this as outlined in Section 8.2.
  - Rescale the probability back to the original scale and update agent probability by taking a gradient step, where the step size is limited by $\min(p_i^{r_1} + p_j^{r_1}, p_i^{r_2} + p_j^{r_2})$.

end

Note that though not necessarily cheaper than exhaustive search, this generalisation to the multi-agent case serves the purpose of providing intuition for our subsequent choices of a considerate cost in the air traffic control application, where there are more than 2 agents in the system.

8.4 Numerical results for multi-agent anti-coordination games

With all the theoretical derivation of ensuring convergence to the social optimum in place, we can run a series of experiments to validate this. We analyse the behaviour of a specific two-player, two-action game, showing the phase portraits and simulated trajectories for a range of k and γ. There are special values of k for which optimising the social welfare objective does not achieve the socially optimum pure strategy. Further we show that the addition of a considerate cost will correct the behaviour for the desired convergence. We also give numerical examples of the multi-action and multi-agent case, we demonstrate that Algorithm 8 and Algorithm 9 successfully find the social optimum in each case. Detailed results can be found in Appendix I.

Chapter 9

Considerate costs for arrival management problem

To recap, one problem that we see in the row-wise optimisation algorithm of Section 6.2.2 is that, due to the simple greedy approach, each aircraft only acts in its own best interest. In the three aircraft example (Figure 6.4), simply increasing the weight on the conflict cost will only result in the second aircraft moving further away, leading to a socially sub-optimal solution. And although increasing the entropy parameter may help shift the last aircraft out of blocking regions, it also makes each aircraft less distinguishable and inferring the Monge map less accurate.

Like the games discussed in Chapter 8, for the air traffic control problem, we are also in an anti-coordination game setting. In order to gain better utility, the agents (planes) should aim to pick different actions (timeslots). From our discussion of externalities and designing custom 'considerate' costs, we want to introduce an analogous cost here to take into consideration the preferences of other aircraft when optimising.

9.1 Considerate cost formulation

Recall in our entropy-regularised set-up, we are able to find a closed-form solution when optimising on each row: $\boldsymbol{p_i^*} = \exp\{-\boldsymbol{k_i}/\epsilon\}/(\mathbb{1}^\intercal \exp\{-\boldsymbol{k_i}/\epsilon\})$, where $\boldsymbol{k_i}$ encapsulates both the conflict cost and the transport cost for the ith aircraft (with the elements defined in (6.10)). What we want to do is further incorporate a term to capture social impact in $\boldsymbol{k_i}$, so that $\boldsymbol{k_i}$ is replaced by $\boldsymbol{k_i} + \gamma \boldsymbol{k_i^s}$, where γ is a parameter that we can choose.

To design a suitable $\boldsymbol{k_i^s}$, let us first consider (entropy aside) how much each aircraft contributes to the total objective cost in their current location(s). That is, for each i, we find the inner product

$$c_i = \langle \boldsymbol{k_i}, \boldsymbol{p_i} \rangle.$$

Next, to capture the preferences, we find the derivative of $R(P) + L(P)$, i.e. the matrix D given by

$$D_{ij} = \frac{\partial(R+L)}{\partial P_{ij}}.$$

The *advantage* of other time slots is given by subtracting the derivative from their current contribution

$$\boldsymbol{c}\mathbb{1}_N^\intercal - D.$$

We call this the advantage since it computes how good each slot is relative to the current slot. There is no need to introduce any cost to the current aircraft i taking a good slot for aircraft l if aircraft l prefers their current allocation.

For each slot we measure the biggest inconvenience to others by taking the maximum advantage over all other aircraft (i.e. all the rows). We also take the positive part, so that the aircraft is not incentivised to place mass at slots where no-one else wants to be (i.e. we assume free disposal as discussed previously). The resulting 'considerate cost' is then given by

$$\boldsymbol{k_i^s} = \max_{\{l:l\neq i\}} \left((\boldsymbol{c}\mathbb{1}_N^\intercal)_{l\cdot} - D_{l\cdot}\right)^+.$$

Finally we add this considerate cost onto $\boldsymbol{k_i}$ and solve the optimisation as before

$$\boldsymbol{p}^* = \frac{\exp\{-(\boldsymbol{k_i} + \gamma \boldsymbol{k_i^s})/\epsilon\}}{\mathbb{1}_n^\intercal \exp\{-(\boldsymbol{k_i} + \gamma \boldsymbol{k_i^s})/\epsilon\}}, \tag{9.1}$$

and update by

$$\boldsymbol{p_i^{(t+1)}} = (1-w)\boldsymbol{p_i^{(t)}} + w\boldsymbol{p^*}. \tag{9.2}$$

To compute the matrix D above, there is an implicit assumption that the current aircraft has not moved, i.e. it is still occupying its current slot. We would like to remove the constraints that the current aircraft is causing on the others before we find their preferred slots. This is equivalent to supposing that the ith aircraft is not contributing to any cost in the objective function, and the effect can be achieved by removing the ith row from the transport cost C and the transport matrix P when computing $\boldsymbol{k_i^s}$.

If $\boldsymbol{k_i} + \gamma \boldsymbol{k_i^s}$ converges for all i, then this algorithm converges. By removing the current aircraft from the system and finding the preferences of the other aircraft, this means the interest of this current individual in occupying each location is balanced compared with other planes who would benefit from taking over these spaces the most.

We also note that the above algorithm has additional computational costs in comparison to the naïve row updates presented earlier. In particular, it is expensive to compute the gradients D. Previously, there was only one such row computation per update. By the formulation above, we would have to compute this for each of the N aircraft, i.e. we are scaling our computation time by a factor of N.

To mitigate this, we note that it is likely the aircraft which will be most affected by the preference of one particular flight are its nearest neighbours. Therefore, we can first find the closest neighbours and then find the gradient of only those aircraft (of course nearest neighbour search also comes with a cost $O(k\log(k))$ for k neighbours unless we incur storage costs by keeping an ordered list).

Algorithm 10: Modified row-wise algorithm with considerate costs (MORAC)

Input: List of arrival times, $\boldsymbol{t} \in \mathbb{R}^n$, transport cost matrix C, discretisation scale, N, relevant parameters for the objective function, and a relaxation parameter w, convergence criterion: objective remains within a tolerance of *tol* within the last T steps.

Convert arrival times to a vector in $\{0,1\}^N$

Initialise the transport matrix as the identity, $\mathbb{I}_{N\times N}$

Reduce the dimension of the problem by removing all rows without an aircraft for the transport and the cost matrix to $\mathbb{R}^{n\times N}$, denote the reduced transport matrix $\tilde{\mathbb{I}}$

Find the number of conflicts, $R(\tilde{\mathbb{I}})$ and objective value $\mathcal{L}$ of the starting configuration.

if $R(\tilde{\mathbb{I}}) > 0$ **then**

  while *Last improvement greater than tolerance is within T steps* **do**

    for *Each row* **do**

      Compute the linear term which captures the conflict and transport cost $\boldsymbol{k_i}$.

      if *only keep neighbours* **then**

        $update_indices = neighbours_indices$

      end

      else

        $update_indices = all$

      end

      Compute the considerate cost by searching over the *update_indices*.

      Compute $\boldsymbol{p^*}$ using (9.1).

      Update this row: $\boldsymbol{p_i^{(t+1)}} \leftarrow (1-w)\boldsymbol{p_i^{(t)}} + w\boldsymbol{p^*}$

      If using temperature, update the visit frequency.

      Choose the next index, either cyclically or by visit frequency.

    end

    Compute and store new objective

    if *All conflicts resolved,* $R(P^{(t+1)}) = 0$ **then**

      break

    end

  end

end

Remark 9.1.1. *If we directly apply Algorithm 7, the next aircraft is chosen in a deterministic, cyclic manner. However, we may wish to take a more randomised approach. We can add a parameter which acts like a temperature (as in simulated annealing) but is localised, so that planes which have been moved and their neighbours are more likely to get updated. For the implementation of this, we draw the next aircraft from the empirical distribution of the visit frequencies. To be precise, we initialise the frequency vector with $\mathbb{1}_n$. To determine our next index, we draw from the normalised distribution inferred from the frequency vector (i.e. multiply the frequency vector by some constant so that we have a discrete distribution in $\mathcal{S}_n$). Once the new index is chosen, we increment that element in the frequency (and that of its immediate neighbours if the neighbours option is chosen) by 1.*

This is an example of Pólya's urn [66]. The probability of choosing each aircraft at the nth iteration, X_i^n, is a martingale. To see this, suppose that $X_n^i = y/z$, then

$$E[X_i^{n+1}] = \frac{y}{z}\frac{y+1}{z+1} + \frac{z-y}{z}\frac{y}{z+1} = \frac{y}{z}\left(\frac{y+1}{z+1} + \frac{z-y}{z+1}\right) = \frac{y}{z}.$$

The limiting distribution of the proportion can be shown to be a uniform $U[0,1]$ *distribution.*

The neighbours and the temperature modifications discussed are optional, we set out the modified row-wise algorithm with considerate cost (MORAC) in Algorithm 10.

9.2 Numerical experiments

In this section we look at several numerical experiments performed using MORAC.

Figures 9.1 and 9.2 give the results of solving the same toy problem as that for Figures 6.2 and 6.3, but this time with considerate costs computed from $k = 5$ neighbours, a local temperature parameter (as described in Remark 9.1.1), and consideration coefficient $\gamma = 0.5$. In this example, the objective is monotone decreasing, though in general this is not guaranteed with the considerate cost algorithm, particularly when we over-relax the updates by having a high w in (9.2) or place a higher emphasis on the considerate cost through γ in (9.1).

The top diagram of Figure 9.2 shows the initial distribution of arrivals. At the bottom of Figure 9.2 we show that all conflicts have been resolved in the suggested output. Here, the performance of MORAC is similar to the performance of the row-wise algorithm (Algorithm 7) in Figures 6.2 and 6.3.

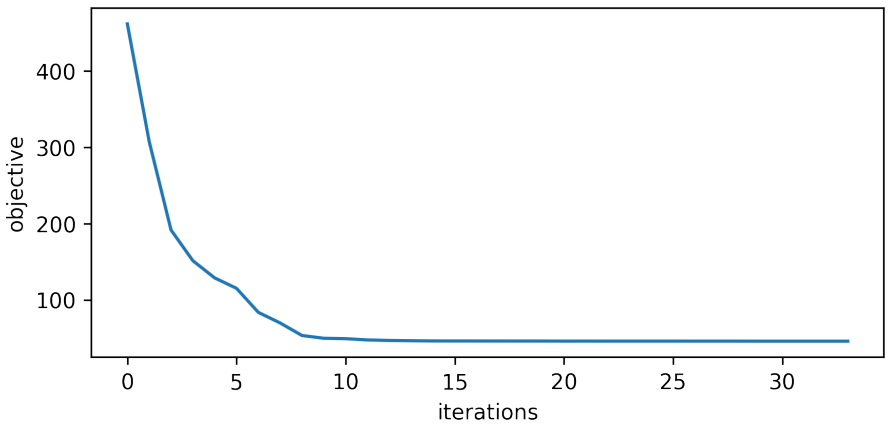


Figure 9.1: An example of the monotonically decreasing objectives for the row-wise algorithm with considerate cost (MORAC).

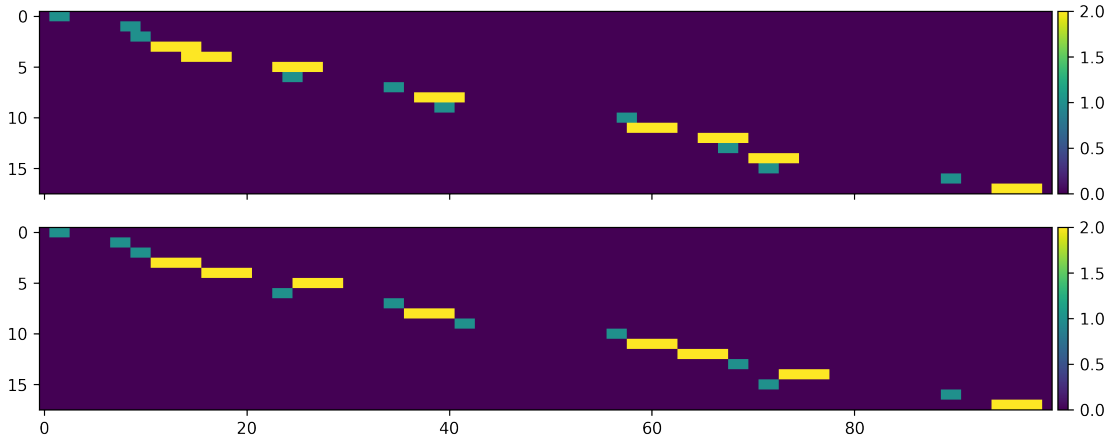


Figure 9.2: An example of application of row-wise algorithm with considerate cost. This uses a cubic transport cost with $c = 1$, $\alpha = 40$, $\beta = 1$, $\epsilon = 1$, $w = 0.5$, $k = 5$, $\gamma = 0.5$, and the minimum separation vector for aircraft of type 1 and 2 is given by $[q(1), q(2)] = [2, 5]$. The top diagram shows the starting distribution and the bottom diagram shows the converged output. The colour of the bars represent the aircraft type and the length of the bars represents separation distance. Overlaps in the columns represents conflicts.

The benefit of adding the considerate cost in MORAC can be seen in Figure 9.3. We show the same three aircraft example that the original row-wise algorithm had issues optimising. As we can see from the top subfigure, only the first two aircraft are in conflict with each other. The third plane has no overlap with either two. As discussed in Section 6.2.2, without considerate costs, there is little incentive for the last aircraft to move since its initial configuration does not have any conflicts. The reason why we observe movement of the last aircraft in the middle diagram is due to the regularisation parameter. However, these conflicts are fully resolved when we add the considerate cost term into the optimisation problem — there is now a cost for the last plane to remain in the same spot, therefore it moves as desired. It is precisely these interaction/opportunity costs we want this additional social/considerate term to capture.

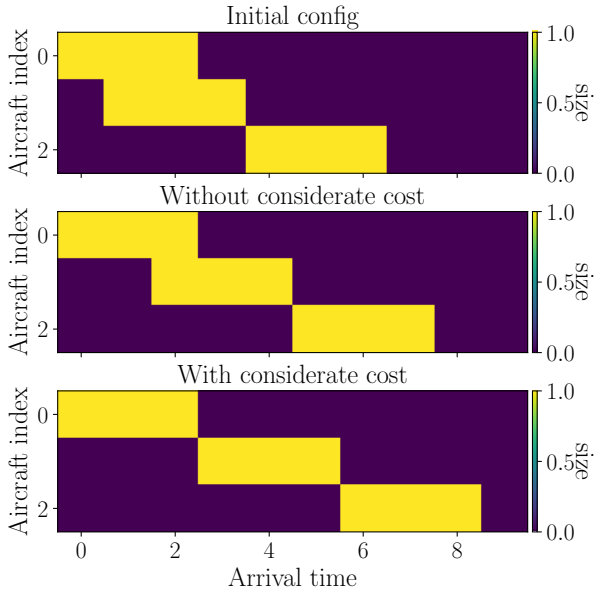


Figure 9.3: A three aircraft system. Top: The initial distribution. The first two aircraft are too close together, but there is little incentive for the last aircraft to move as it does not have any pairwise conflicts with the first two aircraft. Middle: the solution from the row-wise algorithm. Due to the regularisation parameter, the third aircraft has moved slightly but the problem is not resolved. Bottom: with the addition of considerate costs, at the same level of regularisation ϵ we have fully resolved the conflicts.

9.3 Real data example

Finally we test our algorithm on some real data from July 2019 (peak travel time before the pandemic). The AMAN system receives frequent messages (around once per 11 seconds), each message contains information on all planes that are within 500 nautical miles from Heathrow. There is an estimated time of arrival, and AMAN will issue a target time based on the landing sequence as discussed in Section 1.2.1. For a given message time, we can apply our algorithm to this list of estimated arrival times.

Before doing so, let us first note that the set-up of our approach in computing conflict costs described a symmetrical approach — the separation gap was attributed to the aircraft type rather than consecutive pairs of aircraft types. As motivated by Chapter 1, we want similar sized aircraft to land together, so the optimal separation gap is asymmetric. Therefore, we add an additional matrix look-up to infer the minimal gap of each pair when computing the conflict cost; this separation gap matrix is inferred from data. Because the separation gap is time-based, this will depend on the weather conditions on the day, so we reconstruct this information to use in our MORAC algorithm.

To be precise, since AMAN optimises over the sequence order and then maps the sequence into target times based on the configurations of the day (which depends on factors like visibility conditions), we can infer the minimum separation gap for the data sample by looking at the differences in target times of arrival for pairs of successive aircraft. From this, we can construct a matrix of average separation times, Q, where the row index represents the weight category of the leading aircraft, and the column index represents the weight category of the second aircraft. Therefore we introduce an extra matrix look-up for the separation

times. Practically, this means the convolutions in (6.6)–(6.8) would be over $q(s_i, s_{i+1})$ (an element of Q) instead of $q(i)$, where s_i is the weight category of the ith plane.

We introduce two additional features to the transport costs, asymmetry and inhomogeneity. The cost asymmetry captures the fact that it is easier to slow down an aircraft than to speed it up. The inhomogeneity represents the greater flexibility to move aircraft that have arrival times further into the future. We still acknowledge that there is only a range of physical slots to swap to, so a very high cost is set if that range is violated. A representation of the asymmetric cost can be seen in Figure 9.4. In this figure, the y index represents the original time slot of the aircraft, the x index represents the new time slot, and the value represents the transport cost for this swap. The exact values and ranges still requires some input from the domain experts. In Figure 9.5, we see the transport cost when the above is applied to a real data sample and we remove the 'empty' rows as described in MORAC, Algorithm 10 (the new y index represents the plane index).

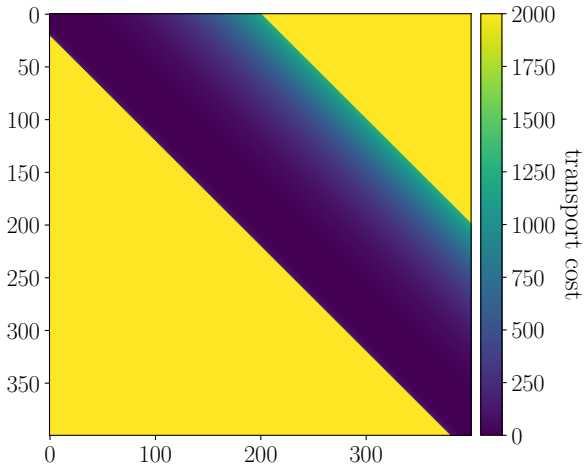


Figure 9.4: Illustration of custom transport cost matrix capturing more realistic considerations. This is a cubic cost with different parameters depending on whether we are requiring the aircraft to move forwards or backwards. Here it is assumed that moving backward is easier than moving forward.

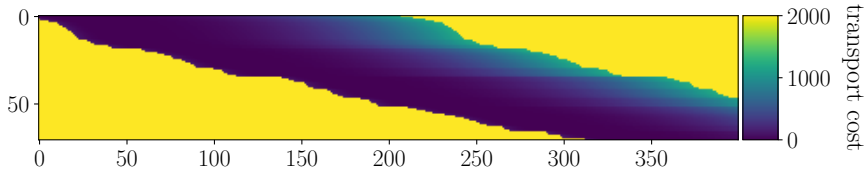


Figure 9.5: The inhomogeneous cost for the data sample with rows removed if there are no aircraft at the index currently.

We set the initial message time to be time 0. We normalise the arrival times to be within [0,1] by dividing through by 1.2 times the final predicted arrival time. This is so that there are some extra slots to move the aircraft to (i.e. the final flights can still be delayed). As the data is prior the pandemic, Heathrow runways used to reach around 98% capacity, hence delays are inevitable.

Let $\alpha = 60$, $\beta = 1$, $\epsilon = 1$, $w = 0.5$, $k = 10$, $\gamma = 0.5$. Let the transport cost be as illustrated in Figure 9.4. We discretise to 400 intervals, and construct the minimum separation gap from the data. In terms of aircraft size, this corresponds to a gap of 5 for medium-medium, 9 for heavy-medium, and 7 for heavy-heavy.

The original data is depicted in the top subfigure of Figure 9.7. This is an extremely busy system, with 71 aircraft needing to be sequenced within 400 intervals. All of the aircraft have a separation distance of at least 5 spaces, hence some conflicts are inevitable.

When we apply the original row-wise algorithm (without considerate cost), the objective values can be seen in Figure 9.6, and the final configuration can be seen in the middle subplot of Figure 9.7. The final objective value is 7029, with 84 conflicts (row-wise overlaps).

The output of the MORAC algorithm (with considerate costs) can be seen in Figure 9.6 and the bottom plot of Figure 9.7. The objective values have converged and returned a final value of 6115 with 87 conflicts.

Despite having marginally higher remaining conflicts, the final objective is much lower than the case without considerate costs. The reason for this can be observed in Figure 9.7, the movement (changes in slots) are far more local in the case with the considerate cost, therefore incurring lower transport costs. Some planes move significantly further away when there are no considerate costs. This resembles the behaviour we observed at the bottom of Figure 6.4, where due to the blocking effect of other aircraft, the middle aircraft had to move further away to reduce conflicts. It is apparent that, in this real data example, there are blocking effects too, which the considerate cost alleviates successfully, finding a much lower objective value.

The difficulty in dealing with real data is that the current system AMAN can set fairly large delays to incoming aircraft. However, because we designed the transport cost such that altering beyond a certain value is infeasible, the algorithm would not introduce as much delay by design. As further work, we can extend the domain and re-design our inhomogeneous transport cost with the help of NATS so that we have feasible optimisation problems.

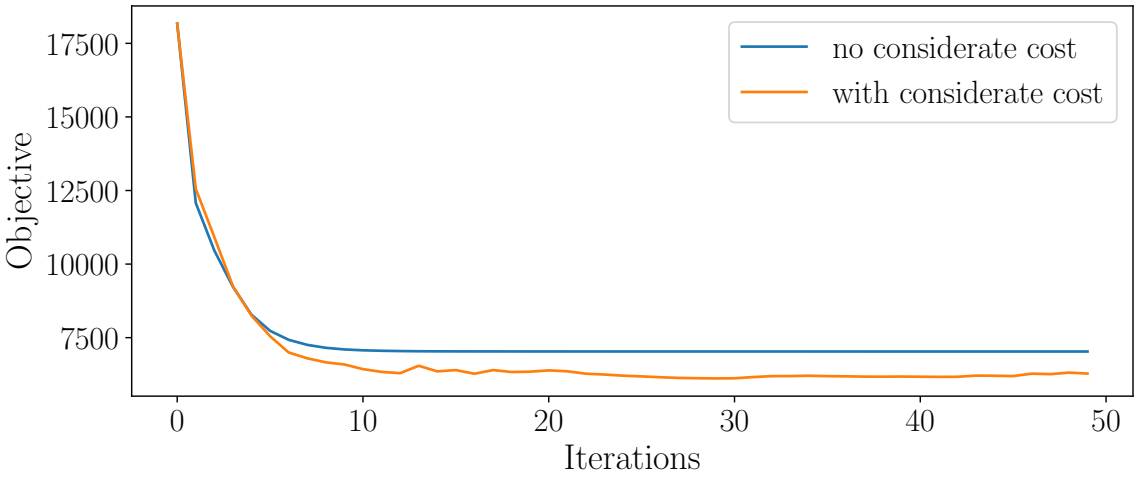


Figure 9.6: Objective functions when we optimise over real data.

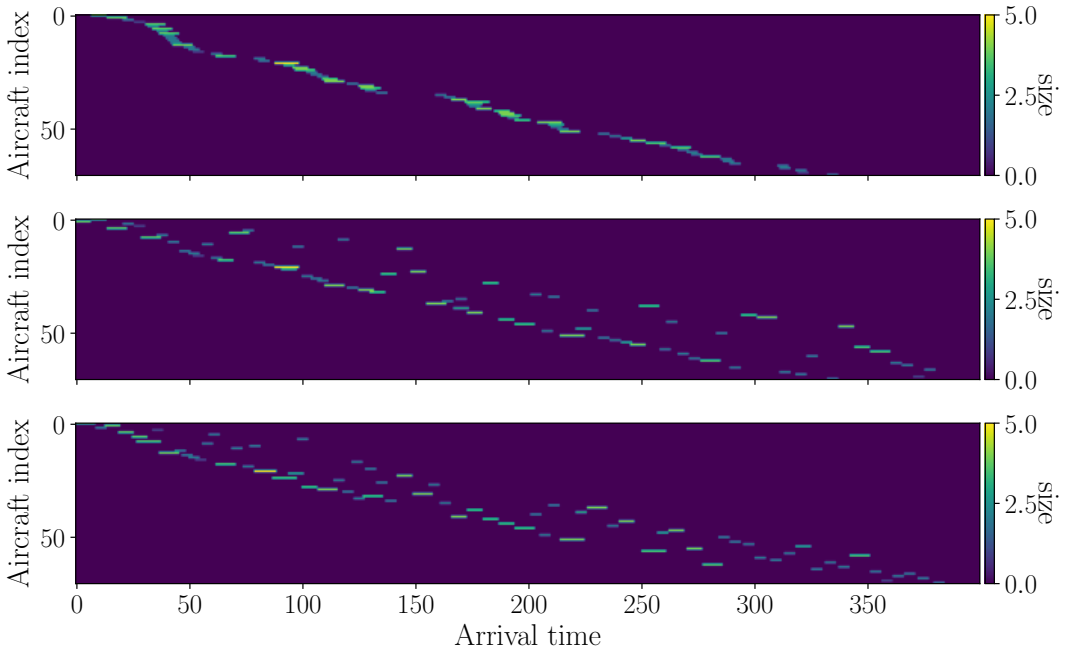


Figure 9.7: Top: The initial configuration of the problem, we see that there are some highly condensed clusters. Middle: the output of the row-wise algorithm without considerate cost. Bottom: the output of MORAC algorithm (with considerate cost).

Chapter 10

Summary

In Part II, we introduced a block coordinate descent-like method with entropic regularisation for arrival management in air traffic control, inspired by ideas from optimal transport and economics. In this problem, the conflict cost is a quadratic term which is not positive definite, and this introduced non-convexity into the objective function.

This method was applied to the arrival management problem by considering each aircraft as a selfish agent wanting to obtain their preferred timeslots whilst minimising conflicts. However, when agents optimise their own utility, this converged to a Nash equilibrium which is not guaranteed to be (locally) Pareto optimal. To compare with a social planner's (non-convex) problem, we derived conditions for local Pareto optimality and saw that optimising the objective functions of individual agents rather than the social objective function corresponds to simply halving the penalty for conflicts.

On a simple three aircraft system, we saw that there may be little incentive for planes to move when they are blocking others, which results in a big increase in costs. Inspired by the theory of externalities in economics, we added an additional 'considerate cost' to our row-wise optimisation, which can alleviate problems like the three aircraft example. This considerate cost is analogous to a Pigouvian tax, and is designed so that the marginal benefit of selfish actions for individual agents is counteracted by considering the marginal cost on others. By performing detailed analysis on a two-agent anti-coordination game (and its extension to more agents), we gained intuition on why adding considerate costs similar to the one we suggested in MORAC (Algorithm 10) may work better and these methods can be more effective than global approaches.

Finally we tested our algorithm on several numerical examples, both toy examples and using real data. Unfortunately, not all conflicts are resolved for the real data. The main reason is because for the data sample, there is a large cluster of aircraft very close to 'arriving' at Heathrow, likely due to the aircraft in the stacks. The very busy nature of Heathrow coupled with the fact that under the current system, delays are accepted/expected at the holds, make comparison of our MORAC method on historical data difficult.

Changing the set-up of the transport costs, and allowing further delays may reduce conflicts further, but this would require domain expert input.

Part III

A decentralised approach

Chapter 11

Arrival streaming over metering points

So far in this thesis, we have focused on the idea of eliminating holding times by optimising the problem at Heathrow. Another key performance indicator for NATS is how many aircraft can achieve what is known as *continuous descent*. This is where an aircraft achieves an *idle thrust descent*, without level restrictions imposed by other aircraft in the system. As a result, there is a lower communication cost and workload on the air traffic controllers as well as increased fuel efficiency. One idea which is being explored at NATS is that of *arrival streaming at metering points*. Metering points are physical locations, and the idea being explored is whether the traffic passing over these points can be managed so that the aircraft are longitudinally separated by time (at each metering point). One motivation for this scheme is that, with adequate longitudinal separation, the aircraft can descend without constraints imposed by proximity to other aircraft, thereby increasing fuel efficiency. Saving fuel is important both environmentally and financially due to tight profit margins (exacerbated by high oil costs).

Recall that arrival management at Heathrow is currently aided by a tool called AMAN (Section 1.2.2). Let us quickly recap the existing AMAN tool. There are two main functionalities of AMAN, the first is to use the available data for *trajectory prediction*, the other is to find a *landing sequence* by optimising the costs as discussed in Section 1.2.2. The current system only cares about the final airport sequence and has no en-route separation in-place. As NATS further explores the arrival streaming over metering points concept, AMAN is being adapted so that the optimisation problem has been broken down into several subproblems, with en-route separation at each metering point. Let us call this *modified AMAN*. It would therefore be interesting to apply the methodologies developed in this thesis (which largely ignore geographic concerns) to this new idea. In this chapter, we look at the model for metering points, and obtain results of applying an adapted version of our algorithm, MORAC, detailed in Chapter 9 to this system.

11.1 Arrival streaming

In this section, we discuss the concept of *arrival streaming* over metering points further. NATS is exploring this as an improvement to the existing AMAN solution. Metering points are a set of waypoints/fixpoints ('fixes') along popular routes, through which air traffic controllers can funnel aircraft, and impose a separation gap of at least 90 seconds between

successive aircraft flying over these points. The process of separation at these metering points is known as *arrival streaming*. The metering points are also known as Time Assigned Points (TAPs).

There are four categories for these metering points, which are referred to as TAP A, TAP B, TAP C, and TAP D. The final one, TAP D, is Heathrow airport, i.e. the destination. TAP C's correspond to the current four holding stacks for Heathrow. The other two categories are points further away from Heathrow, with TAP B's being the closer set, and TAP A's being the furthest. Some directions/routes do not have a TAP B, only TAP A and TAP C. See Figure 11.1 for an illustration of these metering points geographically; note that EGLL is the ICAO name for Heathrow airport.

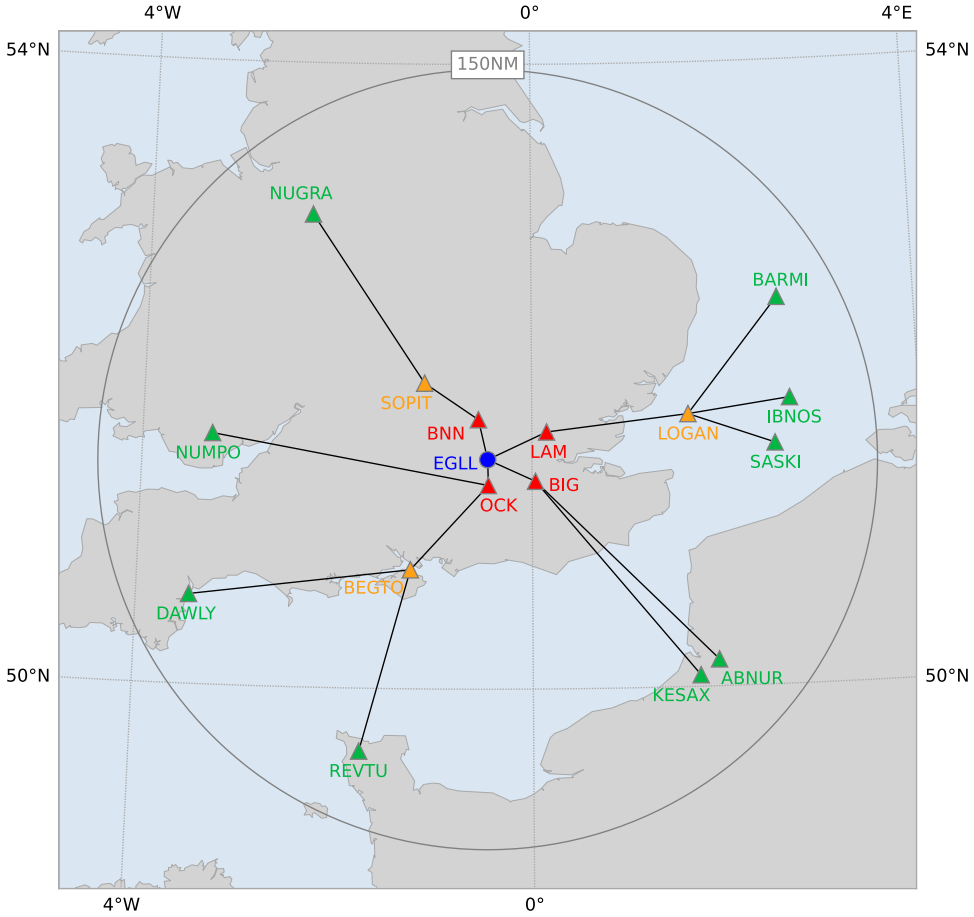


Figure 11.1: Location of metering points. The green points are the TAP A's, the amber points are TAP B's, the red points are TAP C's, and finally the blue point is Heathrow, which is labelled as EGLL — its ICAO name. Note that all metering points are within 150 nautical miles from Heathrow.

11.1.1 Modified AMAN – Under prototyping

For the arrival management on approach to Heathrow, NATS has moved from distance-based separation to time-based separation [9, 10]. The existing AMAN works on distance-based separation, and target times for landing are mapped from internal targets which ensure min-

imum separation distance. As the current system only optimises for separation at Heathrow, it is possible to just change the flight levels (altitude) in order to ensure adequate separation en-route. However, this has the disadvantage of further constraints on aircraft speed/descent on approach to Heathrow, therefore fewer aircraft achieving continuous descent, leading to lower fuel efficiency [40, 169]. Hence the alternative of fixing target times over metering points so that aircraft are streamed as they arrive at metering points seems appealing.

The difference between conventional AMAN and modified AMAN is that the sequence of metering points divides the arrival management problem in modified AMAN into chunks. For the earlier segments (TAPs A, B, and C), as the aircraft are flying at higher altitudes, their sizes do not influence the separation gap at these points, the 90-second rule (discussed above) is fixed for all plane types. Having a time separation of at least 90 seconds is an air traffic rule-of-thumb that ensures the aircraft are separated by at least 5 nautical miles (depending on aircraft speed, it usually corresponds to 7 nautical miles to give controllers a buffer). Any violations of the 90-second rule (by pairs of aircraft) at these TAPs are known as *bunches*.

From the available data, modified AMAN will still use the trajectory prediction functionality to create modified instructions and a target time for aircraft to reach each of their TAPs. This target time depends on the estimated time over metering fix (ETO), the minimum, and the maximum possible time to reach these fixed points based on the aircraft specification. The target time for TAP A's will be calculated and fixed approximately 30 minutes before the aircraft reach TAP A. The target time for TAP B, and TAP C are fixed 5 minutes before the aircraft is predicted to pass its TAP A point.

There is a slight complication for TAP A's due to their location, since flights taking off from nearby airports may not have reached their cruising altitude. The take-off and climbing stage has a lot of uncertainties. For the early versions of modified AMAN, these regional flights were given fixed target times shortly after take-off or whilst still on the ground! There are only very limited actions that can be done for aircraft which have not taken off. Imposing delays on aircraft before take-off is rare, and tends to happen in emergencies only. To address this problem, a target time will only be issued once the aircraft reaches its requested flight level.

As this airspace concept is still in the development phase, the first simple solution is to ignore these regional flights until TAP B and C target times have to be fixed. Due to this issue, the top left subsystem of metering points in Figure 11.1 (starting at NUGRA) only has a single TAP A point feeding into one TAP B point, and then to one TAP C (so that regional flights from e.g. Dublin and Manchester can be handled).

Note that by adding these additional metering points and requirements for separation, the optimisation problem is more complicated when compared with optimising the arrival sequence at the runway. Due to local pop-up flights, and aircraft flying at different speeds, if we solved only the arrival management problem at TAP D (as we have done in Chapters 6 and 9), then this is neither necessary nor sufficient for the separation to be ensured at the earlier TAP points.

There are two modes for experiments with flight simulation at NATS, *streaming* and *delay absorption*. Under the streaming mode, the information from previous fixes are fed forward to the later fixes once decisions have been made on target times. To be more precise, if a target time is issued for TAP A which is different to the current ETO, then it is assumed the aircraft would take corrective action to follow this instruction. As a result, the estimated arrival times, minimum and maximum ranges will be need to be updated, which will have knock-on effects for TAP B and TAP C (their ETOs, minimum and maximum times will

also have to be amended accordingly). Under the delay absorption mode, if an aircraft is predicted to spend time at the holding stacks, TAP C's, then it is possible to absorb some of this delay by introducing speed restrictions at the earlier TAPs.

In the development of this modified AMAN, experiments are currently being performed under the streaming mode. Note that as the delays in the stacks do not matter here, this separates the arrival management problem into 4 disjoint subproblems, one at each holding stack (Tap C). Delay absorption is anticipated to be utilised for future development. One barrier for deploying modified AMAN (which is still under prototyping) on delay absorption is that the system would preferentially delay everything, leading to less room for manoeuvre. To be more precise, a lot of flights may have predicted delays at the holds, and as a result, they would receive a target time that required them to fly at their minimum speed to reach TAP C as late as possible (so that they are held at holding stacks for the minimum time). This means that when new (regional) aircraft pop up in the optimisation problem, e.g. flights taking off from Manchester, the only option would be to slow down new aircraft or speed up others. The system is not robust enough for these situations.

11.2 Modification to the transport problem set-up

Unfortunately, due to the fact that modified AMAN is still under development, a direct comparison with other methods is not possible. We can however demonstrate the performance of our work under this arrival streaming application, and establish a new benchmark. Demonstration of the performance of our algorithm is valuable for NATS as AMAN is costly proprietary software which has long development cycles, and is not amenable to small incremental changes.

As there are now several stages/subproblems to the optimisation problem, modifications have to be made to the optimal transport-inspired algorithm MORAC (Algorithm 10) introduced in Chapter 9 so that it operates on the configuration discussed in Section 11.1.

For example, any aircraft that already have been issued with a target time become 'immovable' (for that particular TAP point), and therefore they do not form part of the optimisation problem that we need to solve, although they still contribute to the collision cost.

The AMOPT algorithm

From Figure 11.1, we see that if we are only optimising TAPs A, B, and C, then the problem from each TAP C can be run in parallel. We adapt our previous algorithm for this setting. The new algorithm for this metering problem, AMOPT, is described in Algorithm 11.

Let the ETO of Tap A for aircraft i be $t_A^{(i)}$. Recall that for each aircraft, the target times need to be fixed at around $t_A^{(i)} - 30$ for Tap A, and at $t_A^{(i)} - 5$ for both the Tap B and C points.

For each message time, we check whether there are any flights that need to be given target times. Since messages come in around once every 11 seconds, in the algorithm, we find any flights within 30.5 minutes or 5.5 minutes of arriving to their Tap A point without an existing target time (depending on whether we are checking Taps A, B, or C). If such flights exist, then we run our row-wise algorithm, MORAC, as described in Chapter 9 (Algorithm 10) to determine the times. It is then assumed these target times would be obeyed by aircraft,

therefore the ETOs, minimum and maximum times for subsequent points also have to be updated as appropriate.

Algorithm 11: Arrival Management with Optimal Transport (AMOPT)

Input : Dataframe containing flight data (ETO, min and max time over each TAP for each aircraft), necessary parameters for the row-wise MORAC algorithm, this includes the design of the cost function, number of intervals to discretise the space to.

for *Each unique message time* **do**
- Modify the minimum and maximum time if needed.
- Filter out regional flights at TAP A.
- **for** *Each TAP A fix* **do**
  - **if** *There are flights with ETO to this TAP A less than 30.5 mins but without a fixed target time* **then**
    - Run the MORAC algorithm to find desired target times for these flights.
    - Fix the new target time(s) for the flight(s)
    - Compute the modified gradients to project ETO, min and max times.
  - **end**
- **end**
- **for** *Each TAP B fix* **do**
  - **if** *There are flights with ETO to this TAP B which is less than 5.5 mins away from ETO of TAP A and does not have a fixed target time* **then**
    - Run the MORAC algorithm to find desired target times.
  - **end**
  - Fix the target time to TAP B of flights
  - Compute the modified gradients to project ETO, min and max times.
- **end**
- **for** *Each TAP C fix* **do**
  - **if** *There are flights with ETO to this TAP B which is less than 5.5 mins away from ETO of TAP A and does not have a fixed target time* **then**
    - Run the MORAC algorithm to find desired target times.
  - **end**
  - Fix the target time to TAP C of flights
- **end**

end

Output: Dataframe with target times for each flight and TAP.

11.3 AMOPT algorithm applied to simulation data

We deploy our adapted algorithm described above for arrival streaming on metering points to simulation data. To be precise, the data are collected in a simulator, and our algorithm is applied ex-post. This is mainly because direct access to the simulator could not be obtained in time. As long as the data contained in messages in the simulator can be pre-processed to be fed into AMOPT and the target times post-processed to be fed back with low latency, the algorithm should be able to run with minimal modifications in the simulator. We note that by design choice, our objective function is different compared to AMAN's and the space that we are optimising over is also different. This was a deliberate choice to give our algorithm

more flexibility compared with modified AMAN.

11.3.1 Scale of the arrival streaming problem

In this section, we give a sense of the size of this arrival streaming problem. In Table 11.1, we see the actual data from NATS for the first 17 days of July 2019 (this is representative of a busy period pre-pandemic). We see the total number of flights passing over the metering points, note that some fixes are busier than others. Recall that not all TAP A points lead to TAP B's, hence why the total number of flights for TAP B points is lower than TAPs A or C totals. Occasionally, flights may be redirected to different TAP C points, and some regional flights may not pass TAP A's, therefore accounting for the higher TAP C total compared with TAP A. In the last column, there is an approximation of how many bunches there were, based on their recorded time over metering points (in AMAN). Note that these also give lower bounds on approximately how many flights could not achieve continuous descent. We can identify the bottlenecks in this system, and which points may benefit the most in terms of reducing the number of bunches.

Metering point	**Number of flights**	**Real world number of flights with $<$ 90 seconds spacing at the fix (as measured by AMAN)**
TAP A	**10913**	**1321**
ABNUR	1827	209
BARMI	586	15
DAWLY	292	10
IBNOS	2308	415
KESAX	217	3
NUGRA	2217	395
NUMPO	940	56
REVTU	818	45
SASKI	1708	173
TAP B	**8098**	**1434**
BEGTO	1171	41
LOGAN	4627	1099
SOPIT	2300	294
TAP C	**11609**	**1590**
BIG	2210	203
BNN	2416	290
LAM	4767	835
OCK	2216	262

Table 11.1: Cumulative flight data from the first 17 days of July 2019. In the second column, we have the total number of flights passing over the metering points. The last column gives a way of measuring the approximate number of bunches, which is also a lower bound on the number of flights for which continuous descent was not possible. Note that extra TAP C flights could come from stack swapping (so the sum of flights going into a TAP C may be higher than the sum of flights from TAP A/B's that feed into it).

Let us now discuss the daily data that AMAN receives. We only receive data for those aircraft that are sufficiently close to Heathrow, this is typically around 500–550 nautical miles or from the point of take-off if this is a regional flight (regional flights start at a distance of around 200 nautical miles or less from Heathrow). At typical speeds, this corresponds to around 60 minutes before the aircraft reach their TAP A fix or 80 minutes before TAP C.

As a result, we typically need to optimise over a period of 80 minutes. Given that the separation time required at each fix is around 90 seconds, this means that any discrete time model would need to have sufficiently fine granularity to capture this.

As mentioned above, updated data arrive to AMAN around once every 11 seconds, at what we call a 'message time'. For each message time, there can be details for up to 20 aircraft. If we merged the information over the course of an hour together in a rolling window, then we may have up to 75 aircraft in the system to optimise over. Note that given separation constraints, this means that some holding at TAP C may be inevitable.

11.3.2 The simulation data

We have one day of simulation data which have the typical traffic patterns as that of before the pandemic aside from fewer regional aircraft. The data consist of 592 flights. In Table 11.2, we see the original number of bunches. Flights passing via NUGRA (over Manchester) create a lot of conflicts in this simulation.

There is an update in the location and predicted arrival time of each aircraft around once every 11 seconds. Although the dataset is from simulation, it is still noisy and there may be jumps in the predicted ETO when the information on an aircraft gets updated, as that is based on the aircraft speed information and radar data. The data were obtained via an advanced simulator at NATS, therefore as well as ETOs, we also have the actual time over (ATO) for each aircraft, which is the final time that they were recorded as flying over the fixes.

Recall that for all aircraft at all times, minimum separation distance has to be maintained for safety which is around 5 nautical miles (a comfortable separation distance for all aircraft weight categories). Since the current practice does not target sequencing at the metering points, vertical separation will be commonly used to achieve adequate/safe spacing in the event where 5 nautical miles longitudinally is not possible. As a result, aircraft can appear in the (simulated) data to arrive at metering points simultaneously but they are actually on different flight levels. The target for our algorithm is to ensure separation in longitude/latitude coordinates, rather than relying on vertical separation.

Note that in real data, as well as having flights at the same coordinates (but different flight levels), there can also be measurement errors in the locations of aircraft, therefore AMAN will record an arrival time for an aircraft as long as the aircraft gets sufficiently close to the fixpoint. One example of where aircraft data can appear 'erroneous' in such dataset is if there is a pair of aircraft that are flying on parallel tracks greater than 5 nautical miles away, when they fly near/over a metering point, their time over these points may be recorded as near identical, despite safety constraints are satisfied.

11.3.3 Updating the minimum and maximum ETO times

Once we fix a target time for TAP A, this will have knock-on effects for the ETOs for TAP B and TAP C, as well as affect the way the minimum and maximum arrival times change. There are two intuitive ways to handle the adjustment of these times.

The first is simply to take a translation of the times by the amount that the ETO for TAP A was adjusted by, that is, we find the increment and add this onto all subsequent ETOs, minimum, and maximum arrival times. This assumes that the aircraft flies at the current speed until TAP A and then immediately switches back to the original speed. This is very simple to compute, but requires keeping track of all the aircraft that had a speed instruction imposed at the 30-minute horizon. Further communication with each of these aircraft once they reach TAP A (to ensure that they revert back to their original speed) may cause additional workload on the air traffic controllers.

One alternative which seems sensible is using linear interpolation given the new target time. A new target time at an earlier fixpoint involves a change in the speed. If we assume the plane carries on flying at the new speed, then the change would affect the later times proportionally. The effect of modifying the times in this way can be seen in Figure 11.2.

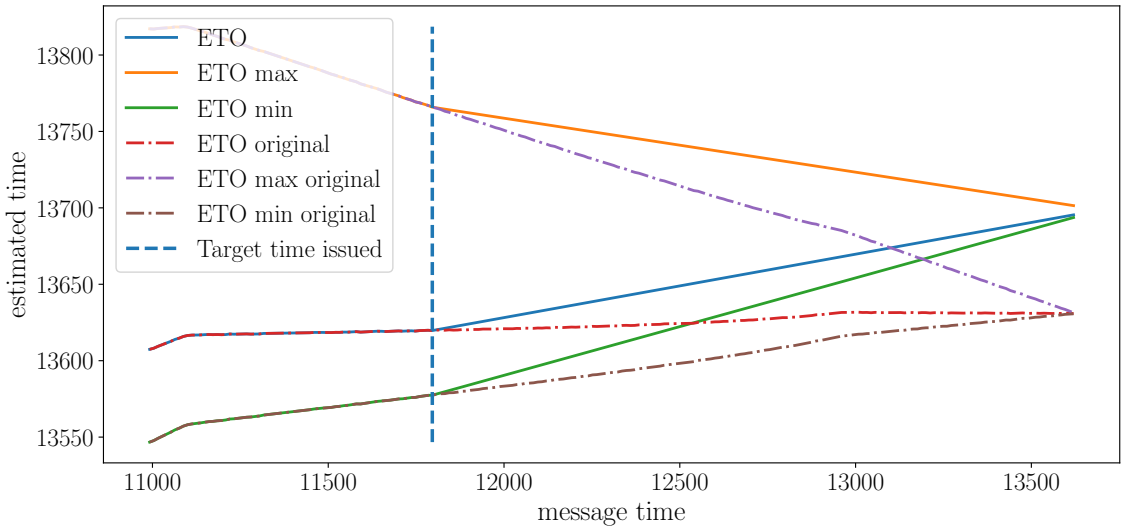


Figure 11.2: Modifiying the ETOs, as well as the ETO min and ETO max by linear interpolation.

11.3.4 The results on simulation data

In this amended arrival streaming problem with metering points, the aircraft are allowed to be held in the stacks (under the preliminary analysis with the streaming mode), and key evaluation metrics are given by:

- The number of bunches, defined as those closer than 90 seconds apart when they pass over the metering points.
- The number of aircraft that cannot have continuous descent due to unresolved bunching for at least one TAP along its route.
- The distribution in change of the target time compared with the original estimated time over (ETO).

TAP type	Fix	Original – ATO	AMOPT
TAP A	ABNUR	16	1
	BARMI	3	0
	DAWLY	0	0
	IBNOS	5	0
	KESAX	0	0
	NUGRA	46	8
	NUMPO	1	0
	REVTU	0	0
	SASKI	5	0
TAP B	BEGTO	2	0
	LOGAN	52	24
	SOPIT	42	12
TAP C	BIG	16	2
	BNN	44	7
	LAM	53	18
	OCK	5	1

Table 11.2: Number of bunches in the metering simulation data before and after applying our algorithm.

Number of bunches

The result of our algorithm, AMOPT, can be seen in Table 11.2. We see that the algorithm has successfully resolved most of the conflicts. We emphasise that having some unsolved bunches is not unsafe, but requires the air traffic controller to separate the aircraft vertically, losing the potential gains that would be provided by continuous descent.

In the case where there are some bunches, we are also interested in the distribution of these separation times, whether it is close to the required 90 seconds or not. In Figure 11.3 we see the histogram of separation times for all the combined bunches listed in Table 11.2. We observe that a lot of flights are very close to the 90 second spacing requirement, and this is a good result for NATS. The decision on the best course of action for these aircraft would be left to the controller.

Some provisional results for modified AMAN ran on a different simulation sample can be seen in Table 11.3. Unfortunately, as modified AMAN is still under development, a direct comparison of AMOPT and modified AMAN has not been possible at this stage (due to the time limits of this DPhil project).

Number of aircraft unable to achieve continuous descent

It may be the case that one flight is involved in multiple bunches. From the bunches results discussed above, we can find the unique callsigns (identifiers for individual aircraft) involved across all these bunches, in order to identify the aircraft which cannot achieve continuous descent. Since we have 4 separate subsystems as defined by their TAP C point, we can list the number of flights unable to achieve continuous descent by these subsystems (Table 11.4). We see that that the number of flights prevented from continuous descent has decreased significantly.

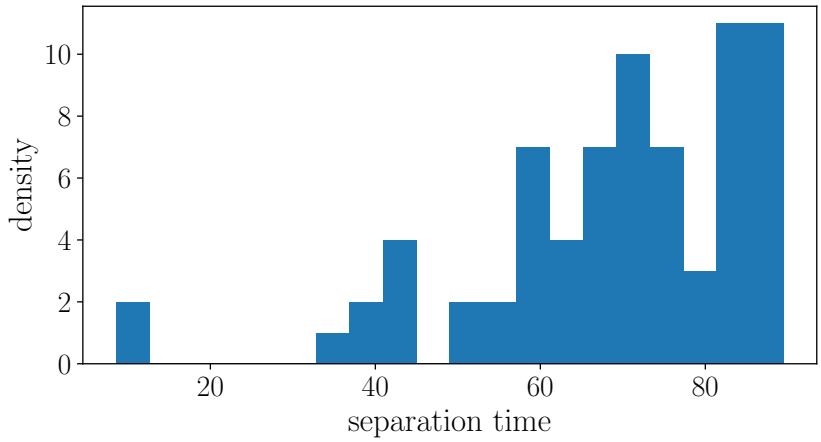


Figure 11.3: Histogram of bunches

Difference in net arrival times

Finally we plot the histogram of the 'disturbance' caused by giving a target time. This is defined to be

$$\text{target time} - \text{ETO at the time of issuing the target time.}$$

The reason why this is of interest is that we don't want to be, on average, adding delays to all aircraft. Also smaller perturbations to their original time may more preferable and fairer compared with if some planes are greatly delayed. Note though that sometimes, leaving some aircraft undisturbed can be preferable in order to reduce the number of instructions that have to be made and the workload on the air traffic controllers. This issue is delicate and the results have to be interpreted with this in mind. In Figure 11.4 we see the histogram of differences for all target times. We see that for most aircraft only minor corrections to their original ETOs are made.

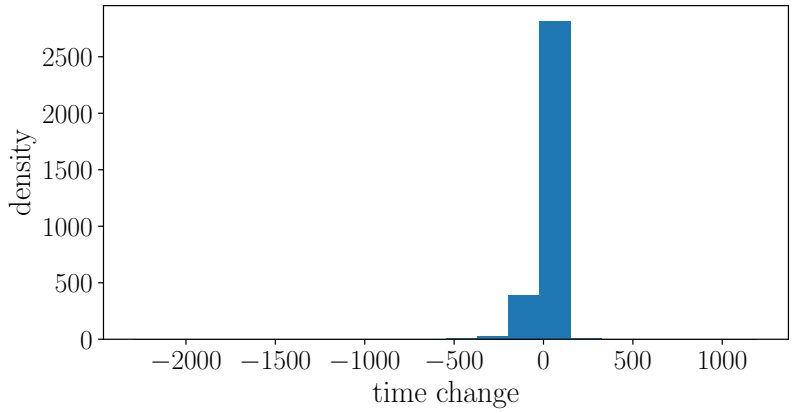


Figure 11.4: Histogram of time changes (difference between target times and ETOs).

TAP type	Fix	Original – ATO	Modified AMAN
TAP A	ABNUR	20	8
	BARMI	2	0
	DAWLY	1	1
	IBNOS	13	7
	KESAX	0	0
	NUGRA	54	23
	NUMPO	1	1
	REVTU	0	0
	SASKI	13	6
TAP B	BEGTO	3	0
	LOGAN	77	28
	SOPIT	49	13
TAP C	BIG	20	7
	BNN	49	14
	LAM	71	21
	OCK	7	3

Table 11.3: Number of bunches when modified AMAN is applied on some alternative settings.

TAP type	Fix	Original – ATO	AMOPT
TAP C	BIG	18	1
	BNN	53	21
	LAM	54	15
	OCK	7	0

Table 11.4: Number of flights prevented from continuous descent in the metering example.

11.4 Discussion

The AMOPT algorithm has successfully removed most bunches from most fixes. Aside from its performance, another important aspect is the formulation and abstraction of the metering problem. AMAN is a piece of costly proprietary software, which takes many years to develop. By setting up the model and algorithm as we have done, we provide many moving pieces that can be manipulated. Easy to change components include:

- The transport cost matrix. We have used a simple cubic function based on the difference between target time and ETO, this can be easily changed and for domain knowledge to be added.
- Size of separation gap at each TAP.
- The separation gap by aircraft type. It is possible to encode the size of the aircraft so that there are different separation gaps for each pair. As a result, this can be naturally extended to handle TAP D (Heathrow).

Therefore the algorithm is highly customisable which can be useful for testing future airspace design ideas for research and development. For example, what happens if we re-

moved all the TAP B points? The configuration and simulation can be done in-house rather than relying on external resources.

Further complications that we have not considered here include considering weather data from the Met Office. The uncertainty of wind conditions means that some theoretical times that we looked at above may be unachievable. For this reason, issuing speed instructions cannot be something trivially constructed from the target time. Other factors may also impact the model, these include how accurate the aircraft speed measurements are, and how quickly the instruction for a target time can be acted upon.

The process of slowdown can also be sensitive for business, the Airbus A320 can be slowed down more than the Boeing B787. How can we ensure fairness in assigning delays? How many aircraft should we move (change the target times of) and how do we decide which are best to be moved?

In this exposition, it is assumed that there will still be some holding. The experiments are currently in the streaming mode, and further work is required to handle the system as a whole (from Heathrow) under the delay absorption mode.

Ultimately we are interested in conflict resolution at the endpoint, which is Heathrow. Does this system of metering points detract from the ultimate goal of minimising delays in the holding stacks? For further work, it would be worth investigating how much gain there is, in optimising for continuous descent if it is at the cost of longer hold delays. A proper fuel burn analysis with some aircraft performance model such as the BADA model [138] should be performed to determine the efficacy of this approach.

11.5 Summary

In this chapter, we looked at how to adapt our algorithm from the previous part of the thesis to tackle a new airspace design concept which is being explored by NATS.

We see that our algorithm, AMOPT, successfully resolves most bunches for most fixes in the system. AMOPT not only deconflicts at these metering points, but has easy to adapt parts that would be amenable to further research and input from domain experts.

Unfortunately, direct comparison of AMOPT and modified AMAN could not be done within the scope of this DPhil. As future work, it would be of interest to perform proper benchmarking by comparing the results of AMOPT against modified AMAN for the same simulation data.

Chapter 12

Conclusion and extensions

In this thesis, we have looked at framing the arrival management problem at Heathrow in several ways. The application of these techniques is novel in the air traffic control industry. In this final chapter of the thesis, we conclude by discussing and looking at the possibilities for future development.

12.1 Summary of the thesis

To summarise, we studied the arrival management problem in air traffic control, in particular, managing capacity when resources are constrained by runway availability at Heathrow airport. Additional considerations come from respecting the different separation distances based on weight categories of aircraft and efficient/practical instructions. We have broken down the arrival management problem into concise, digestible elements that were modelled with two main approaches.

The focus of Part I was to use methods from control theory and reinforcement learning. For this, we designed an environment and looked only at the radial distance from the airport (mapping the problem down to one dimension). We explored several set-ups such as only issuing instructions at two control points, or limiting the frequency of actions.

We note that, due to the choice of discretisation for the state and action space, the number of possible states increases exponentially with discretisation scale, making classic reinforcement learning methods with table representation impractical. Therefore function approximators are needed. We utilised the DDPG algorithm and explored using path signatures as a novel feature set, as well as neural network architectures inspired by geometric multigrid methods.

Using deep reinforcement learning methods come with convergence and stability issues which we analysed. Popular and well-known deep RL methods use a range of handcrafted features, but many also utilise regularisation to mitigate stability issues. This is still an active area of research.

Although reinforcement models can take a long time to train, deploying the model does not take long and can be done in real time if this method was to be adapted for execution. However, it is key to note that in this part of the thesis, our experiments were done in simulation mode, by imposing our own model for aircraft arrivals. This was not validated on any real data. As such, this approach would still require further work before it would become practically implementable, especially in a safety critical system like air traffic control.

In Part II we developed this idea of regularisation and formed a general optimisation

algorithm inspired by ideas from optimal transport and block coordinate descent. By using entropy regularisation (a strongly convex function), we essentially 'convexify' our problem. With some insights on externalities from economic theory, we designed a 'consideration' cost for the arrival management problem and we showed that by optimising with this additional cost in the algorithm MORAC, we achieved better empirical performance on a real data sample. With this approach, we successfully ran our algorithms on several data samples from NATS (in contrast with Part I). Whilst this was done with static data, by making minor modification to the data input functions, this method can be performed dynamically within new arrival management systems.

Our second formulation can also be adapted to impose further structure into the arrival management problem. In Part III, we extended the algorithm MORAC, introduced in Part II, to give target times at metering points when further constraints are imposed. We called this new extended algorithm AMOPT. Due to access issues to the simulator, the results of this part was obtained by running AMOPT on (static) data which had been collected from an advanced simulator at NATS.

Simultaneously a modified AMAN was being tested at NATS and we showed comparable level of performance with AMOPT. With further domain knowledge, it is anticipated that the performance can be improved further. Due to time constraints, a direct comparison of AMOPT and modified AMAN could not be performed.

We note that by design choice, our objective function is different compared to (modified) AMAN's and the space that we are optimising over is also different. This was a deliberate choice to give our algorithm more flexibility compared with modified AMAN.

Another important issue is that AMAN is a piece of costly proprietary software, which takes many years to develop. By setting up the model and algorithm as we have done, we provide many moving pieces that can be modified and improved. Our algorithm is highly customisable which can be useful for testing future airspace design ideas for research and development.

In terms of deployability, as long as the data contained in messages in the simulator can be pre-processed to be fed into AMOPT and the target times post-processed to be fed back with low latency, the algorithm should be able to run dynamically with minimal modifications in the simulator for further testing.

The simulator that we are referring to does not have to be the existing one at NATS, alternative/improved simulators can be utilised in a similar fashion. As these simulators are designed to be as close to real-life situations as possible, and is used for training air traffic controllers, adopting AMOPT in real-life should not cause any additional burden.

12.2 Extensions

One could extend the work of this thesis further. First, one could design an environment to simulate dynamics of aircraft at metering points, so that we can adapt the ideas discussed in Part I to the arrival streaming concept discussed in Part III. In this thesis, we discussed the idea of single-agent reinforcement learning by modelling the air traffic controller as the agent. We have also looked at ideas from multi-player games, where we thought of each aircraft as the agent. A natural blend of these ideas is that of multi-agent reinforcement learning.

12.2.1 Reinforcement learning for arrival streaming

In Part I, the controller was modelled as the agent, choosing actions for each discrete interval, which had an effect on any aircraft that are in those intervals.

Under the streaming idea (Part III), different control points, known as TAPs join/merge and the arrival management problem is distributed across these points. However, target times can only be issued either 30 minutes before reaching TAP A (for the TAP A target times) or 5 minutes before reaching TAP A (for the TAP B and C target times). This is similar to the ideas discussed in Section 3.3.3, where there were two control points at which the reinforcement learning agent issued actions.

We briefly sketch one way that we can formulate an environment for streaming over metering points. As there are 9 TAP A points, let our state be a matrix in $\mathbb{R}^{9\times N}$, where N is the number of intervals used to discretise time. As we modelled in Part I, let the value of the element represent the size/weight category of the aircraft (if there are any in those slots).

Let the action be a vector in $\mathbb{R}^{17}$ (corresponding to the number of TAPs), this represents the target time issued to each TAP point. To be precise, each element of this action will act upon a space representing when a target time should be issued (i.e. the 30 minutes or 5 minutes before arrival to TAP A points). This is a similar extension of the work in Part I, and essentially this is only considering the radial distance away, hence there may be complications with multiple aircraft 'occupying' the same interval at the same time. In that case, we may want to multiply the number of rows in the state and the dimension of the action space, by an appropriate number, to deal with any multiplicity.

The environment describes the arrival and movement of the aircraft. The target times issued to each aircraft can be stored in a dictionary (as a hidden part of the environment that the agent does not directly observe), and the reward gained would be computed from this dictionary. For each instance where there are target times that are too close (i.e. a bunch), a negative reward would be gained. Perhaps an additional reward can be gained if the aircraft achieves continuous descent.

It would then be interesting to use reinforcement learning methods (potentially utilising signature features as discuss in Section 3.3.4) for the streaming problem, and compare with the results of Part III and future results from modified AMAN.

12.2.2 Multi-agent reinforcement learning (MARL)

An interpretation of our row-wise transport algorithm (Chapter 6) from the perspective of social games and externalities motivated the addition of the 'considerate cost' term in our MORAC algorithm (Chapter 9). Game theory ideas have inspired and overlapped with multi-agent reinforcement learning (MARL) algorithms. Indeed deep reinforcement learning techniques have been applied, with great success, to cooperative multi-agent systems.

Blending the ideas from Parts I and II, we can model each aircraft as an agent in a multi-agent system. Of course, similar issues to those discussed in Part I relating to increase in dimension and convergence still persist in the multi-agent case. Modelling multiple agents comes with other problems. In a cooperative, centralised set-up, the action space is usually formulated as a joint action across all the agents. Hence the action space scales exponentially with the number of agents, making fully centralised set-ups impractical.

Alternatively, we can model this as a decentralised system. Decentralised systems assume that only local information, e.g. observations and action history, are available to be used to

inform an action for a particular agent. Often there is little to no communication between agents. Since each agent acts and optimises for themselves (independently of the others), this would mitigate the issue of exploding state and action spaces common in cooperative, centralised systems. However, training agents individually is difficult, as the actions of the other agents become part of the environment (making observations quite noisy) and such a system may not converge [193]. There are recent work to provide some guarantees for inhomogeneous agents under a decentralised system with function approximation [201, 200]. If there is a table representation, QD-learning [100] can also be applied.

There is another approach gaining traction. Although the agents are acting in/executed on a decentralised system, the training can still be done through a centralised way. This is usually referred to as 'centralised training with decentralised execution' (CTDE) in the literature [151, 36]. In this setting, two of the key issues are how to deal with the exponential growth in states with the number of agents, and how to map the policy from centralised learning into something that can be executed in a decentralised system.

One popular assumption to make for CTDE is that the joint (action) value function can be decomposed/factored. Value decomposition networks [172] use the idea that the joint action value function can be decomposed as the sum of individual action value functions. QMIX [151, 150, 152], another value-based method, builds on this idea by enforcing a monotonicity constraint instead to allow nonlinear relationships between the individual action value functions and the joint action value function. The monotonic relationships ensure consistency is preserved which allows individuals to select the actions based on their own preferences (decentralised execution) whilst preferences would still respect the joint value function. MAVEN [120] uses a combination of value and policy-based ideas to improve exploration through the use of a shared latent variable. There are other representation methods too, e.g. [121]. See [36] for an in-depth analysis of factor decomposition methods, including the impact of the number of factored components.

The above approaches would be interesting to explore for the air traffic control problem. At any one time, we can have a different number of aircraft in the system (which will largely be periodic across days). A lot of MARL methods focus on optimising performance for a fixed number of agents to limit the size of the state space. For air traffic control, local configurations, actions, and conflict resolution methods should be similar which means the primary interest is likely in pairwise/small-group interactions of the agents. This means that we want to learn local behaviours and patterns. Effectively, we are saying we want to mod out some symmetry in subgroups of agents, hence research on learning the similarities across tasks with different numbers of agents can be leveraged ([93]).

12.3 Future airspace research

In this thesis, we modelled the arrival management problem in several ways; these ideas are new to applications in air traffic control, as both reinforcement learning and optimal transport have become more widely adopted in applications only in the last few years.

We have developed ideas that are extensible, with some potential approaches highlighted above. We discussed in Part III of this thesis that there are plenty of components in our AMOPT algorithm which can be modified, like adding domain knowledge for the cost of rearranging aircraft. However, there are many other possibilities for further work. For example, some airlines may choose to pay extra for prioritised slots, in which case we can look at ideas from auction design to determine a fair pricing structure, and queuing theory

to model how such a system would play out.

In collaboration with the Alan Turing Institute, NATS will be making further advances in novel techniques and applications in air traffic control under *Project Bluebird* [6] over the next few years. One goal of the project is to create a digital twin of UK airspace – an accurate, data-informed, real-time simulator. The aim is to recreate the real-life system as much as possible. This would make open source research in air traffic control applications easier and more accessible. We can utilise the digital twin to see how the work from this thesis can benefit the overall framework. As we have looked specifically at arrivals to Heathrow, the simulator would quantify the benefit brought to the whole air traffic control system, that is, showing how freeing up capacity in arrival management would impact the overall airspace. We touched briefly on the concept of fairness in Part III (how 'best' to assign target times leading to delays), and NATS will look at similar issues as part of Bluebird via AI ethics.

Finally, another key component of this project is to further utilise deep reinforcement learning for the automation of some tasks within air traffic control, and explore the robustness of non-linear function approximators. The tools and techniques that we examined and developed in this thesis can contribute to this space.

Bibliography

[1] ACI World releases preliminary 2017 world airport traffic rankings Passenger traffic: Indian and Chinese airports major contributors to growth Air cargo: Volumes surge at major hubs as trade wars threaten. https://aci.aero/2018/04/09/aci-world-releases-preliminary-2017-world-airport-traffic-rankings-passenger-traffic-indian-and-chinese-airports-major-contributors-to-growth-air-cargo-volumes-surge-at-major-hubs-as-trade-wars-thre/. Retrieved: 22nd March 2022.

[2] Global sepsis alliance. https://www.global-sepsis-alliance.org/sepsis. Retrieved: 9th July 2021.

[3] Making better use of our existing runways. https://www.heathrow.com/content/dam/heathrow/web/common/documents/company/about/consultation/Making%20Better%20use%20of%20our%20existing%20runways.pdf. Retrieved: 24th April 2022.

[4] Nats: About us. https://www.nats.aero/about-us/. Retrieved: 19th May 2022.

[5] NEWS2 Early Warning Score. https://www.rcplondon.ac.uk/projects/outputs/national-early-warning-score-news-2. Retrieved: 17th September 2021.

[6] Project Bluebird: An AI system for air traffic control. https://www.turing.ac.uk/research/research-projects/project-bluebird-ai-system-air-traffic-control. Retrieved: 15th May 2022.

[7] Sesar: Delivering the digital European sky. https://www.sesarju.eu/. Retrieved: 19th May 2022.

[8] SKYbrary, index of articles relating to events that included wake vortex turbulance as a contributory factor. https://www.skybrary.aero/index.php/Accident_and_Serious_Incident_Reports:_WAKE. Retrieved: 15th September 2021.

[9] (2018). Enhanced Time Based Separation adds valuable resilience to Heathrow operation. https://www.nats.aero/news/enhanced-time-based-separation-adds-valuable-resilience-heathrow-operation/. Retrieved: 19th May 2022.

[10] (2020). Time-based separation at Heathrow: Optimised airport resilience and reducing delays. https://www.airport-technology.com/features/time-based-separation-at-heathrow-optimised-airport-resilience-and-reducing-delays/. Retrieved: 19th May 2022.

[11] Agarwal, A., Kakade, S. M., Lee, J. D., and Mahajan, G. (2021). On the Theory of Policy Gradient Methods: Optimality, Approximation, and Distribution Shift. *Journal of Machine Learning Research*, 22(98):1–76.

[12] Aitken, A. C. (1950). Studies in practical mathematics. on the iterative solution of a system of linear equations. *Proceedings of the Royal Society of Edinburgh*, 63:652–656.

[13] Amari, S. (1998). Natural Gradient Works Efficiently in Learning. *Neural Computation*, 10(2):251–276.

[14] Arrow, J. (1951). An Extension of the Basic Theorems of Classical Welfare Economics. In Neyman, J., editor, *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pages 507–532. University of California Press.

[15] Arrow, K. J. and Debreu, G. (1954). Existence of an Equilibrium for a Competitive Economy. *Econometrica*, 22(3):265–290.

[16] Barles, G. and Souganidis, P. (1991). Convergence of approximation schemes for fully nonlinear second order equations. *Asymptotic Analysis*, 4(3):271–283.

[17] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2017). Automatic Differentiation in Machine Learning: A Survey. *J. Mach. Learn. Res.*, 18(1):5595–5637.

[18] Bellman, R. E. (1957). *Dynamic Programming*. Dover Publications, Inc., USA.

[19] Bergin, J. and MacLeod, W. B. (1993). Continuous Time Repeated Games. *International Economic Review*, 34(1):21–37.

[20] Bertsekas, D. (2017). *Dynamic Programming and Optimal Control: Volume I*. Number v. 1 in Athena scientific optimization and computation series. Athena Scientific, 4th edition.

[21] Bertsekas, D. and Tsitsiklis, J. (1996). *Neuro-Dynamic Programming*. Athena Scientific.

[22] Bertsekas, D. P. (1992). Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications*, 1(1):7–66.

[23] Bertsekas, D. P. and Castanon, D. A. (1989). The auction algorithm for the transportation problem. *Annals of Operations Research*, 20(1):67–96.

[24] Bhandari, J. and Russo, D. (2019). Global Optimality Guarantees For Policy Gradient Methods. *arXiv*, abs/1906.01786.

[25] Bhandari, J. and Russo, D. (2021). On the Linear Convergence of Policy Gradient Methods for Finite MDPs. In Banerjee, A. and Fukumizu, K., editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 2386–2394. PMLR.

[26] Bianco, L., Dell'Olmo, P., and Giordani, S. (1999). Minimizing total completion time subject to release dates and sequence-dependent processing times. *Annals of Operations Research*, 86:393–415.

[27] Bonneel, N., van de Panne, M., Paris, S., and Heidrich, W. (2011). Displacement Interpolation Using Lagrangian Mass Transport. *ACM Trans. Graph.*, 30(6):1–12.

[28] Brandt, A. and Livne, O. E. (2011). *Multigrid techniques: 1984 guide with applications to fluid dynamics*. Classics in applied mathematics ; vol 67. Society for Industrial and Applied Mathematics, Philadelphia, Pa., revised edition.

[29] Brenier, Y. (1991). Polar factorization and monotone rearrangement of vector-valued functions. *Communications on Pure and Applied Mathematics*, 44(4):375–417.

[30] Briggs, W., Henson, V., and McCormick, S. (2000). *A Multigrid Tutorial: Second Edition*. Other Titles in Applied Mathematics. Society for Industrial and Applied Mathematics.

[31] Brittain, M. and Wei, P. (2019). Autonomous Air Traffic Controller: A Deep Multi-Agent Reinforcement Learning Approach. *arXiv*, abs/1905.01303.

[32] Browder, F. E. (1967). Existence and perturbation theorems for nonlinear maximal monotone operators in Banach spaces. *Bulletin of the American Mathematical Society*, 73(3):322–327.

[33] Burden, R. and Faires, J. (2010). *Numerical Analysis*. Cengage Learning, 9th edition.

[34] Calamai, P. H. and Moré, J. J. (1987). Projected gradient methods for linearly constrained problems. *Mathematical Programming*, 39(1):93–116.

[35] Cartea, A., Perez Arribas, I., and Sánchez-Betancourt, L. (2020). Optimal execution of foreign securities: A double-execution problem. *SSRN*, abstract=3562251.

[36] Castellini, J., Oliehoek, F. A., Savani, R., and Whiteson, S. (2021). Analysing factorizations of action-value networks for cooperative multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 35(2):25.

[37] Cen, S., Cheng, C., Chen, Y., Wei, Y., and Chi, Y. (2020). Fast Global Convergence of Natural Policy Gradient Methods with Entropy Regularization. *arXiv*, abs/2007.06558.

[38] Cesa-Bianchi, N., Gentile, C., Lugosi, G., and Neu, G. (2017). Boltzmann Exploration Done Right. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6287–6296, Red Hook, NY, USA. Curran Associates Inc.

[39] Chang, B., Meng, L., Haber, E., Ruthotto, L., Begert, D., and Holtham, E. (2018). Reversible Architectures for Arbitrarily Deep Residual Neural Networks. *arXiv*, abs/1709.03698.

[40] Chen, H. and Solak, S. (2016). Value of Extended Time-Based Metering for Optimized Profile Descent-Based Arrival Operations. *Transportation Research Record*, 2600(1):27–38.

[41] Chen, K.-T. (1958). Integration of Paths–A Faithful Representation of Paths by Noncommutative Formal Power Series. *Transactions of the American Mathematical Society*, 89(2):395–407.

[42] Chen, R. T. Q., Rubanova, Y., Bettencourt, J., and Duvenaud, D. K. (2018). Neural Ordinary Differential Equations. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc.

[43] Chevyrev, I. and Kormilitzin, A. (2016). A Primer on the Signature Method in Machine Learning. *arXiv*, abs/1603.03788.

[44] Chipman, J. S. and Tian, G. (2012). Detrimental externalities, pollution rights, and the "Coase theorem". *Economic Theory*, 49(2):309–327.

[45] Clarke, A. (2013). How slowing down flights can make them more efficient. https://nats.aero/blog/2013/07/how-slowing-down-flights-can-make-them-more-efficient/. Retrieved: 15th September 2021.

[46] Coase, R. H. (1960). The Problem of Social Cost. *The Journal of Law & Economics*, 3:1–44.

[47] Cochrane, T., Foster, P., Chhabra, V., Lemercier, M., Salvi, C., and Lyons, T. (2021). SK-Tree: a systematic malware detection algorithm on streaming trees via the signature kernel. *arXiv*, abs/2102.07904.

[48] Cohen, S. and Elliott, R. (2015). *Stochastic Calculus and Applications*. Probability and Its Applications. Springer New York, 2nd edition.

[49] Cohen, S., Foster, J., Foster, P., Lou, H., Lyons, T., Morley, S., Morrill, J., Ni, H., Palmer, E., Wang, B., Wu, Y., Yang, L., and Yang, W. (2022). Subtle variation of sepsis-III definitions influences predictive performance of machine learning. *Under review*.

[50] Cominetti, R. and Martín, J. S. (1994). Asymptotic analysis of the exponential penalty trajectory in linear programming. *Mathematical Programming*, 67(1):169–187.

[51] Consortium, D. (2022). Empirical comparison of NEW2 and supervised learning methods to predict deterioration for COVID-19 patients. *In preparation*.

[52] Courty, N., Flamary, R., Tuia, D., and Rakotomamonjy, A. (2017). Optimal Transport for Domain Adaptation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(9):1853–1865.

[53] Couzin, I. D., Krause, J., James, R., Ruxton, G. D., and Franks, N. R. (2002). Collective memory and spatial sorting in animal groups. *Journal of Theoretical Biology*, 218(1):1–11.

[54] Crandall, M. G., Ishii, H., and Lions, P.-L. (1992). User's guide to viscosity solutions of second order partial differential equations. *Bulletin of the American Mathematical Society*, 27:1–67.

[55] Crandall, M. G. and Lions, P.-L. (1983). Viscosity Solutions of Hamilton–Jacobi Equations. *Transactions of the American Mathematical Society*, 277(1):1–42.

[56] Cuchiero, C., Larsson, M., and Teichmann, J. (2019). Deep neural networks, generic universal interpolation, and controlled ODEs. *arXiv*, abs/1908.07838.

[57] Cuturi, M. (2013). Sinkhorn Distances: Lightspeed Computation of Optimal Transport. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc.

[58] Dear, R. G. (1976). *The Dynamic Scheduling of Aircraft in the Near Terminal Area*. PhD Thesis, MIT, Cambridge.

[59] Debreu, G. (1951). The Coefficient of Resource Utilization. *Econometrica*, 19(3):273–292.

[60] Degris, T., White, M., and Sutton, R. S. (2012a). Off-policy actor-critic. In *Proceedings of the 29th International Conference on Machine Learning*, ICML'12, pages 179–186, Madison, WI, USA. Omnipress.

[61] Degris, T., White, M., and Sutton, R. S. (2012b). Off-Policy Actor-Critic. *arXiv*, abs/1205.4839.

[62] Dessein, A., Papadakis, N., and Deledalle, C.-A. (2017). Parameter estimation in finite mixture models by regularized optimal transport: a unified framework for hard and soft clustering. *arXiv*, abs/1711.04366.

[63] Di Pietro, D. A., Hülsemann, F., Matalon, P., Mycek, P., Rüde, U., and Ruiz, D. (2021). Towards robust, fast solutions of elliptic equations on complex domains through hybrid high-order discretizations and non-nested multigrid methods. *International Journal for Numerical Methods in Engineering*, 122(22):6576–6595.

[64] Dulac-Arnold, G., Evans, R., Sunehag, P., and Coppin, B. (2015). Deep Reinforcement Learning in Large Discrete Action Spaces. *arXiv*, abs/1512.07679.

[65] E, W. (2017). A Proposal on Machine Learning via Dynamical Systems. *Communications in Mathematics and Statistics*, 5(1):1–11.

[66] Eggenberger, F. and Pólya, G. (1923). Über die Statistik verketteter Vorgänge. *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, 3(4):279–289.

[67] Falgout, R. (2006). An introduction to algebraic multigrid. *Computing in Science Engineering*, 8(6):24–33.

[68] Fedorenko, R. P. (1962). A relaxation method for solving elliptic difference equations. *USSR Computational Mathematics and Mathematical Physics*, 1(4):1092–1096.

[69] Fedorenko, R. P. (1964). The speed of convergence of one iterative process. *USSR Computational Mathematics and Mathematical Physics*, 4(3):227–235.

[70] Forsyth, P. A. and Labahn, G. (2007). Numerical methods for controlled Hamilton–Jacobi–Bellman PDEs in finance. *Journal of Computational Finance*.

[71] Franklin, J. and Lorenz, J. (1989). On the scaling of multidimensional matrices. *Linear Algebra and its Applications*, 114-115:717–735.

[72] Frey, B. S. (1997). A Constitution for Knaves Crowds out Civic Virtues. *The Economic Journal*, 107(443):1043–1053.

[73] Gangbo, W. and McCann, R. J. (1996). The Geometry of Optimal Transportation. *Acta Mathematica*, 177(2):113–161.

[74] Gaspar, F. J., Lisbona, F. J., and Clavero, C. (2000). Multigrid Methods and Finite Difference Schemes for 2D Singularly Perturbed Problems. In *Revised Papers from the Second International Conference on Numerical Analysis and Its Applications*, NAA '00, pages 316–324, Berlin, Heidelberg. Springer-Verlag.

[75] Geist, M., Scherrer, B., and Pietquin, O. (2019). A Theory of Regularized Markov Decision Processes. *arXiv*, abs/1901.11275.

[76] Gifford Jr., A. and Stone, C. C. (1973). Externalities, Liability and the Coase Theorem:A mathematical analysis. *Economic Inquiry*, 11(3):260–269.

[77] Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering*. Applications of mathematics : Stochastic modelling and Applied probability. Springer.

[78] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Sardinia, Italy. PMLR.

[79] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[80] Graham, B. (2013). Sparse arrays of signatures for online character recognition. *arXiv*, abs/1308.0371.

[81] Grondman, I., Busoniu, L., Lopes, G. A. D., and Babuska, R. (2012). A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307.

[82] Hackbusch, W. (1985). *Multi-grid methods and applications*. Springer series in computational mathematics ; Vol 4. Springer-Verlag, Berlin.

[83] Hambly, B. and Lyons, T. (2010). Uniqueness for the signature of a path of bounded variation and the reduced path group. *Annals of Mathematics*, 171(1):109–167.

[84] Hamill, L. and Gilbert, N. (2015). *Agent-based modelling in economics*. John Wiley & Sons.

[85] Han, D. and Wan, J. W. L. (2013). Multigrid Methods for Second Order Hamilton–Jacobi–Bellman and Hamilton–Jacobi–Bellman–Isaacs Equations. *SIAM Journal on Scientific Computing*, 35(5):S323–S344.

[86] Harsanyi, J. C. (1973). Oddness of the number of equilibrium points: A new proof. *International Journal of Game Theory*, 2(1):235–250.

[87] Haskins, P. (2013). Infographic: Why do aircraft hold at Heathrow Airport. https://nats.aero/blog/2013/06/infographic-why-do-aircraft-hold-at-heathrow-airport/. Retrieved: 17th September 2021.

[88] He, J. and Xu, J. (2019). MgNet: A unified framework of multigrid and convolutional neural network. *Science China Mathematics*, 62(7):1331–1354.

[89] Hirshleifer, J. (1983). From weakest-link to best-shot: The voluntary provision of public goods. *Public Choice*, 41(3):371–386.

[90] Hoertel, N., Blachier, M., Blanco, C., Olfson, M., Massetti, M., Rico, M. S., Limosin, F., and Leleu, H. (2020). A stochastic agent-based model of the sars-cov-2 epidemic in france. *Nature Medicine*, 26(9):1417–1421.

[91] Howard, R. A. (1960). *Dynamic Programming and Markov Processes*. MIT Press, Cambridge, MA.

[92] Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 448–456. JMLR.org.

[93] Iqbal, S., De Witt, C. A. S., Peng, B., Boehmer, W., Whiteson, S., and Sha, F. (2021). Randomized entity-wise factorization for multi-agent reinforcement learning. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 4596–4606. PMLR.

[94] Jaakkola, T., Jordan, M., and Singh, S. (1993). Convergence of Stochastic Iterative Dynamic Programming Algorithms. In Cowan, J., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems*, volume 6. Morgan-Kaufmann.

[95] Johnson, A. E., Pollard, T. J., Shen, L., Lehman, L.-W. H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Anthony Celi, L., and Mark, R. G. (2016). MIMIC-III, a freely accessible critical care database. *Scientific Data*, 3(1):160035.

[96] Kakade, S. M. (2001). A Natural Policy Gradient. In Dietterich, T., Becker, S., and Ghahramani, Z., editors, *Advances in Neural Information Processing Systems*, volume 14. MIT Press.

[97] Kalsi, J., Lyons, T., and Arribas, I. P. (2020). Optimal execution with rough path signatures. *SIAM Journal on Financial Mathematics*, 11(2):470–493.

[98] Kamnitsas, K., Ledig, C., Newcombe, V. F. J., Simpson, J. P., Kane, A. D., Menon, D. K., Rueckert, D., and Glocker, B. (2017). Efficient multi-scale 3D CNN with fully connected CRF for accurate brain lesion segmentation. *Medical Image Analysis*, 36:61–78.

[99] Kantorovich, L. V. (2006). On the Translocation of Masses. *Journal of Mathematical Sciences*, 133(4):1381–1382.

[100] Kar, S., Moura, J. M. F., and Poor, H. V. (2013). $\mathcal{QD}$-Learning: A Collaborative Distributed Strategy for Multi-Agent Reinforcement Learning Through consensus + innovations. *IEEE Transactions on Signal Processing*, 61(7):1848–1862.

[101] Katrutsa, A., Daulbaev, T., and Oseledets, I. (2017). Deep Multigrid: learning prolongation and restriction matrices. *arXiv*, abs/1711.03825.

[102] Ke, T.-W., Maire, M., and Yu, S. X. (2017). Multigrid Neural Architectures. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4067–4075.

[103] Kendall, M. and Gibbons, J. (1990). *Rank Correlation Methods*. Charles Griffin Book Series. Edward Arnold.

[104] Kidger, P., Morrill, J., Foster, J., and Lyons, T. (2020). Neural Controlled Differential Equations for Irregular Time Series. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 6696–6707. Curran Associates, Inc.

[105] Kiefer, J. and Wolfowitz, J. (1952). Stochastic Estimation of the Maximum of a Regression Function. *The Annals of Mathematical Statistics*, 23(3):462–466.

[106] Kingma, D. P. and Ba, J. (2015). Adam: A Method for Stochastic Optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA*.

[107] Kohl, N. and Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *IEEE International Conference on Robotics and Automation, 2004, Proceedings. ICRA '04.*, volume 3, pages 2619–2624.

[108] Koutsoupias, E. and Papadimitriou, C. (2009). Worst-case equilibria. *Computer Science Review*, 3(2):65–69.

[109] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc.

[110] Krylov, N. (1980). *Controlled Diffusion Processes*. Stochastic Modelling and Applied Probability. Springer Berlin Heidelberg.

[111] Kuhn, H. W. (1955). The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97.

[112] Lemercier, M., Salvi, C., Damoulas, T., Bonilla, E., and Lyons, T. (2021). Distribution Regression for Sequential Data. In Banerjee, A. and Fukumizu, K., editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 3754–3762. PMLR.

[113] Lévy, B. and Schwindt, E. L. (2018). Notions of optimal transport theory and how to implement them on a computer. *Computers & Graphics*, 72:135–148.

[114] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico*.

[115] Lin, M., Chen, Q., and Yan, S. (2013). Network In Network. *arXiv*, abs/1312.4400.

[116] Liu, C., Zhu, L., and Belkin, M. (2020). Loss landscapes and optimization in over-parameterized non-linear systems and neural networks. *arXiv*, abs/2003.00307.

[117] Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440.

[118] Lyons, T. (2014). Rough paths, Signatures and the modelling of functions on streams. *arXiv*, abs/1405.4537.

[119] Lyons, T. J., Caruana, M., and Lévy, T. (2007). *Differential equations driven by rough paths*. Springer.

[120] Mahajan, A., Rashid, T., Samvelyan, M., and Whiteson, S. (2019). MAVEN: Multi-Agent Variational Exploration. *arXiv*.

[121] Mahajan, A., Samvelyan, M., Mao, L., Makoviychuk, V., Garg, A., Kossaifi, J., Whiteson, S., Zhu, Y., and Anandkumar, A. (2021). Tesseract: Tensorised actors for multi-agent reinforcement learning. In Meila, M. and Zhang, T., editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 7301–7312. PMLR.

[122] Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA. PMLR.

[123] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

[124] Monge, G. (1781). *Mémoire sur la théorie des déblais et des remblais*. De l'Imprimerie Royale.

[125] Morrill, J., Kidger, P., Yang, L., and Lyons, T. (2021). Neural Controlled Differential Equations for Online Prediction Tasks. *arXiv*, abs/2106.11028.

[126] Morrill, J., Salvi, C., Kidger, P., Foster, J., and Lyons, T. (2020a). Neural Rough Differential Equations for Long Time Series. *arXiv*, abs/2009.08295.

[127] Morrill, J. H., Kormilitzin, A., Nevado-Holgado, A. J., Swaminathan, S., Howison, S. D., and Lyons, T. J. (2020b). Utilization of the Signature Method to Identify the Early Onset of Sepsis From Multivariate Physiological Time Series in Critical Care Monitoring. *Critical care medicine*, 48(10):e976–e981.

[128] Mueller, D. C. (2003). *Public Choice III*. Cambridge University Press, 3rd edition.

[129] Nash, J. (1951). Non-Cooperative Games. *Annals of Mathematics*, 54(2):286–295.

[130] Nash, J. F. (1950). Equilibrium points in n-person games. *Proceedings of the National Academy of Sciences*, 36(1):48–49.

[131] Negishi, T. (1960). Welfare economics and existence of an equilibrium for a competitive economy. *Metroeconomica*, 12(2-3):92–97.

[132] Negishi, T. (1961). Monopolistic Competition and General Equilibrium. *The Review of Economic Studies*, 28(3):196–201.

[133] Nenna, L. and Pass, B. (2019). Variational problems involving unequal dimensional optimal transport. *arXiv*, abs/1904.00939.

[134] Nesterov, Y. (2004). *Introductory lectures on convex optimization.* Applied Optimization, vol 87. Springer.

[135] Ni, H., Szpruch, L., Sabate-Vidales, M., Xiao, B., Wiese, M., and Liao, S. (2021). Sig-Wasserstein GANs for Time Series Generation. *arXiv*, abs/2111.01207.

[136] Nisan, N., Roughgarden, T., Éva Tardos, and Vazirani, V. V. (2007). *Algorithmic Game Theory.* Cambridge University Press, New York, NY, USA.

[137] Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization.* Springer New York, NY, 2nd edition.

[138] Nuic, A., Poles, D., and Mouillet, V. (2010). BADA: An advanced aircraft performance model for present and future ATM systems. *International Journal of Adaptive Control and Signal Processing*, 24(10):850–866.

[139] Øksendal, B. (2010). *Stochastic Differential Equations: An Introduction with Applications.* Universitext. Springer Berlin Heidelberg, 6th edition.

[140] Patrascu, A. and Necoara, I. (2015). Efficient random coordinate descent algorithms for large-scale structured nonconvex optimization. *Journal of Global Optimization*, 61:19–46.

[141] Peyré, G. and Cuturi, M. (2018). Computational Optimal Transport. *arXiv*, abs/1803.00567.

[142] Pham, D.-T., Tran, N. P., Alam, S., Duong, V., and Delahaye, D. (2019). A Machine Learning Approach for Conflict Resolution in Dense Traffic Scenarios with Uncertainties. In *ATM 2019, 13th USA/Europe Air Traffic Management Research and Development Seminar*, Online proceedings at http://www.atmseminar.org, Vienne, Austria.

[143] Pham, H. (2009). *Continuous-Time Stochastic Control and Optimization with Financial Applications.* Springer, 1st edition.

[144] Pigou, A. C. (1924). *The economics of welfare.* London : Macmillan, 2nd edition.

[145] Pong, V., Gu, S., Dalal, M., and Levine, S. (2018). Temporal Difference Models: Model-Free Deep RL for Model-Based Control. *arXiv*, abs/1802.09081.

[146] Powell, M. J. D. (1973). On search directions for minimization algorithms. *Mathematical Programming*, 4:193–201.

[147] Powell, W. B. and Ma, J. (2011). A review of stochastic algorithms with continuous value function approximation and some new approximate policy iteration algorithms for multidimensional continuous applications. *Journal of Control Theory and Applications*, 9(3):336–352.

[148] Psaraftis, H., N. (1978). A Dynamic Programming approach to the Aircraft Sequencing problem. Flight Transportation Laboratory report 206481, MIT, Cambridge.

[149] Puterman, M. L. and Shin, M. C. (1978). Modified Policy Iteration Algorithms for Discounted Markov Decision Problems. *Management Science*, 24(11):1127–1137.

[150] Rashid, T., Farquhar, G., Peng, B., and Whiteson, S. (2020a). Weighted QMIX: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv*.

[151] Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. (2018). QMIX: Monotonic value function factorisation for deep multi-agent reinforcement learning. *arXiv*.

[152] Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J., and Whiteson, S. (2020b). Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(1).

[153] Recht, B. (2019). A Tour of Reinforcement Learning: The View from Continuous Control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):253–279.

[154] Reizenstein, J. (2017). Calculation of Iterated-Integral Signatures and Log Signatures. *arXiv*, abs/1712.02757.

[155] Reyna, M. A., Josef, C. S., Jeter, R., Shashikumar, S. P., Westover, M. B., Nemati, S., Clifford, G. D., and Sharma, A. (2020). Early Prediction of Sepsis From Clinical Data: The PhysioNet/Computing in Cardiology Challenge 2019. *Critical Care Medicine*, 48(2).

[156] Robbins, H. and Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407.

[157] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation. In Navab, N., Hornegger, J., Wells, W. M., and Frangi, A. F., editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham. Springer International Publishing.

[158] Rothschild, E. (1994). Adam Smith and the Invisible Hand. *The American Economic Review*, 84(2):319–322.

[159] Salvi, C., Cass, T., Foster, J., Lyons, T., and Yang, W. (2021). The Signature Kernel Is the Solution of a Goursat PDE. *SIAM Journal on Mathematics of Data Science*, 3(3):873–899.

[160] Samuelson, P. A. (1954). The Pure Theory of Public Expenditure. *The Review of Economics and Statistics*, 36(4):387–389.

[161] Schmitzer, B. (2016). A Sparse Multiscale Algorithm for Dense Optimal Transport. *Journal of Mathematical Imaging and Vision*, 56(2):238–259.

[162] Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. (2015). Trust Region Policy Optimization. *arXiv*, abs/1502.05477.

[163] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. *arXiv*, abs/1707.06347.

[164] Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T., Leach, M., Kavukcuoglu, K., Graepel, T., and Hassabis, D. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489.

[165] Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., and Riedmiller, M. (2014). Deterministic Policy Gradient Algorithms. In Xing, E. P. and Jebara, T., editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 387–395, Bejing, China. PMLR.

[166] Simonyan, K. and Zisserman, A. (2015). Very Deep Convolutional Networks for Large-Scale Image Recognition. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA*.

[167] Singh, S., Jaakkola, T., Littman, M. L., and Szepesvári, C. (2000). Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. *Machine Learning*, 38(3):287–308.

[168] Sinkhorn, R. (1964). A Relationship Between Arbitrary Positive Matrices and Doubly Stochastic Matrices. *The Annals of Mathematical Statistics*, 35(2):876–879.

[169] Solak, S. and Chen, H. (2017). Optimal Metering Point Configurations for Optimized Profile Descent Based Arrival Operations at Airports. *Transportation Science*, 52(1):150—170.

[170] Southwell, R. V. (2010). *Relaxation Methods in Engineering Science: A Treatise on Approximate Computation*. Pickard Press.

[171] Strang, G. (2007). *Computational science and engineering*. Wellesley–Cambridge Press, Wellesley, MA.

[172] Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., Lanctot, M., Sonnerat, N., Leibo, J. Z., Tuyls, K., and Graepel, T. (2017). Value-decomposition networks for cooperative multi-agent learning. *arXiv*.

[173] Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press, 2nd edition.

[174] Sutton, R. S., Maei, H., and Szepesvári, C. (2008). A Convergent O(n) Temporal-difference Algorithm for Off-policy Learning with Linear Function Approximation. In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, *Advances in Neural Information Processing Systems*, volume 21. Curran Associates, Inc.

[175] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy Gradient Methods for Reinforcement Learning with Function Approximation. In Solla, S., Leen, T., and Müller, K., editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press.

[176] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9.

[177] Toratani, D. (2019). Application of merging optimization to an arrival manager algorithm considering trajectory-based operations. *Transportation Research Part C: Emerging Technologies*, 109:40–59.

[178] Tsitsiklis, J. and Van Roy, B. (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, 42(5):674–690.

[179] Tsitsiklis, J. N. (1994). Asynchronous stochastic approximation and Q-learning. *Machine Learning*, 16(3):185–202.

[180] van Hasselt, H. (2012). Reinforcement Learning in Continuous State and Action Spaces. In Wiering, M. and van Otterlo, M., editors, *Reinforcement Learning: State-of-the-Art*, pages 207–251. Springer Berlin Heidelberg, Berlin, Heidelberg.

[181] Vickrey, W. (1961). Counterspeculation, acutions, and competitive sealed tenders. *The Journal of Finance*, 16(1):8–37.

[182] Villani, C. (2008). *Optimal Transport: Old and New*. Grundlehren der mathematischen Wissenschaften. Springer Berlin Heidelberg.

[183] Wächter, A. and Biegler, L. T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106(1):25–57.

[184] Wahl, L. M. and Nowak, M. A. (1999). The Continuous Prisoner's Dilemma: I. Linear Reactive Strategies. *Journal of Theoretical Biology*, 200(3):307–321.

[185] Walsh III, J. D. and Dieci, L. (2019). A real-valued auction algorithm for optimal transport. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 12(6):514–533.

[186] Wang, L., Cai, Q., Yang, Z., and Wang, Z. (2019). Neural Policy Gradient Methods: Global Optimality and Rates of Convergence. *arXiv*, abs/1909.01150.

[187] Wang, W., Á, M., and Carreira-Perpiñán (2013). Projection onto the probability simplex: An efficient algorithm with a simple proof, and an application. *arXiv*, abs/1309.1541.

[188] Watkins, C. J. (1989). *Learning from delayed rewards*. PhD Thesis, University of Cambridge.

[189] Wesseling, P. (1992). *An introduction to multigrid methods*. Pure and applied mathematics. Wiley, Chichester.

[190] Williams, R. J. and Baird, L. C. (1993). Analysis of Some Incremental Variants of Policy Iteration: First Steps Toward Understanding Actor–Critic Learning Systems. Technical Report Report NU-CCS-93-11, College of Computer Science, Northeastern University, Boston, MA.

[191] Wilson, R. (1971). Computing Equilibria of N-Person Games. *SIAM Journal on Applied Mathematics*, 21(1):80–87.

[192] Wright, S. J. (2015). Coordinate descent algorithms. *Mathematical Programming*, 151(1):3–34.

[193] Wunder, M., Littman, M., and Babes, M. (2010). Classes of multiagent q-learning dynamics with ϵ-greedy exploration. ICML'10, pages 1167–1174, Madison, WI, USA. Omnipress.

[194] Xu, Y. and Yin, W. (2017). A Globally Convergent Algorithm for Nonconvex Optimization Based on Block Coordinate Update. *Journal of Scientific Computing*, 72(2):700–734.

[195] Yavneh, I. (2006). Why Multigrid Methods Are So Efficient. *Computing in Science and Engineering*, 8(6):12–22.

[196] Yong, J. and Zhou, X. (1999). *Stochastic Controls: Hamiltonian Systems and HJB Equations*. Stochastic Modelling and Applied Probability. Springer New York.

[197] Yserentant, H. (1993). Old and new convergence proofs for multigrid methods. *Acta Numerica*, 2:285–326.

[198] Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. In Fleet, D., Pajdla, T., Schiele, B., and Tuytelaars, T., editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham. Springer International Publishing.

[199] Zeiler, M. D., Krishnan, D., Taylor, G. W., and Fergus, R. (2010). Deconvolutional networks. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2528–2535.

[200] Zhang, K., Yang, Z., and Başar, T. (2019). Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. *arXiv*, abs/1911.10635.

[201] Zhang, K., Yang, Z., Liu, H., Zhang, T., and Başar, T. (2018). Fully Decentralized Multi-Agent Reinforcement Learning with Networked Agents. *arXiv*, abs/1802.08757.

Appendix A

Dynamic programming

In this appendix, we give a brief overview of discrete dynamic programming, and describe value iteration and policy iteration — two very important classes of methods.

Dynamic programming (DP) assumes full knowledge of the Markov decision process and is said to be a *model-based* method. This model-based method uses the model dynamics $p(s', r|s, a)$ as described in (2.1), and other representations given by (2.2)–(2.4) to determine the optimal policy.

The dynamic programming principle (DPP) involves breaking down the problem into subproblems. The state and action value functions, denoted V and Q respectively, essentially store the solutions of these subproblems. To solve the prediction problem, where we want to find the state values under a policy π, we can use (2.9) as an update rule. For each $s \in \mathcal{S}$, the kth iteration is given by

$$V_k(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma V_{k-1}(s')]. \tag{A.1}$$

Clearly V_π is a fixed point to this update, in fact we have $V_k \to V_\pi$ as $k \to \infty$. Finding V_π by fixed point iteration is known as *policy evaluation*.

Two well known DP algorithms are policy iteration and value iteration. We can split policy iteration into two main parts, *policy evaluation* and *policy improvement*. Policy evaluation is the prediction problem discussed above to find the state values V_k. Policy improvement usually involves choosing a policy π that is greedy with respect to the updated state values, that is, the π which optimises the right hand side of (A.1) for a fixed V_{k-1}. We then iterate between these two stages until the policy converges. The pseudocode is given in Algorithm 12. Note that in this formulation the state values are updated *in-place*, this means that as soon as $V(s_i)$ has been updated, it may be used in the calculation of $V(s_j)$ (akin to updates in Gauss–Seidel method).

The problem with policy iteration is that we need to wait until the policy evaluation has converged (within a tolerance) before making a policy improvement. This can be a slow process. If we only update each state value once in the policy evaluation phase, we obtain value iteration, and the update can written as

$$\begin{aligned} V_{k+1}(s) &= \max_a Q_k(s, a) \\ &= \max_a E_\pi[R_t + \gamma V_k(S_{t+1})|S_t = s, A_t = a] \\ &= \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma V_k(s')]. \end{aligned} \tag{A.2}$$

Algorithm 12: Policy Iteration

1. Initialisation

Parameters: a small threshold $\theta > 0$ determining accuracy of estimation (convergence criterion).

Initialize $V(s)$, and $\pi(s) \in \mathcal{A}$ arbitrarily for all $s \in S$.

Initialize $\Delta(i) > \theta$ for all i in $\{1, 2, \ldots, |S|\}$.

2. Policy Evaluation

repeat

  for *each* $s \in \mathcal{S}$ **do**

    $V \leftarrow V(s)$,

    $V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s)) [r + \gamma V(s')]$,

    $\Delta(s) \leftarrow |V - V(s)|$.

  end

until $\max(\Delta) < \theta$;

3. Policy Improvement

$PolicyStable \leftarrow True$

for *each* $s \in \mathcal{S}$ **do**

  $A \leftarrow \pi(s)$,

  $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$,

  If $A \neq \pi(s)$, then $PolicyStable \leftarrow False$.

end

If *PolicyStable*, then stop and return $V \approx V^*$ and $\pi \approx \pi^*$; else go back to Step 2.

This update rule compresses the policy evaluation and improvement into one step. See Algorithm 13 for details.

Generalised policy iteration is a family of methods that alternate between policy evaluation and policy improvement. At one extreme we have policy iteration, which waits until policy evaluation converges before improving the policy. At the other extreme we have value iteration which only takes one update of the values before improvement. Other methods in the family can stop at some point in between. Note that the updates of each iteration in (A.1) and (A.2) relies on other estimated state values (the $V_k(s')$) which could be biased. The process of using other estimated values to improve our current estimate is known as *bootstrapping*.

DP performs a wide search of all actions and all states in a one-step transition. This is an example of a synchronous method. Synchronous methods update all state values in turn for each iteration. This means DP methods suffer from the curse of dimensionality and therefore these models are expensive to compute when the number of states/actions is large.

Asynchronous methods can alleviate some of the problems. These methods typically have an ordering to the update of states. Examples of asynchronous algorithms include: prioritised sweeping and real-time DP [173]. In the case of real-time DP, a trajectory is sampled from the model and states are updated as they are visited. Asynchronous methods still converge if all states are updated infinitely often [21].

Algorithm 13: Value Iteration

Parameters: a small threshold $\theta > 0$ determining accuracy of estimation (convergence criterion).
Initialize $V(s)$, for all $s \in S$, arbitrarily except that V(terminal) $= 0$.
Initialize $\Delta(i) > \theta$ for all i in $\{1, 2, \ldots, |S|\}$.
repeat
  for *each* $s \in \mathcal{S}$ **do**
    $V \leftarrow V(s)$,
    $V(s) \leftarrow \max_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$,
    $\Delta(s) \leftarrow |V - V(s)|$.
  end
until $\max(\Delta) < \theta$;
Output a deterministic policy $\pi \approx \pi^*$, such that
$\pi(s) = \text{argmax}_a \sum_{s',r} p(s', r|s, a) [r + \gamma V(s')]$.

Appendix B

Optimal Control Theory

The theory presented in Section 2.1 is driven mainly by the computer science community for solving discrete problems; much of this parallels the theory of optimal control – its continuous cousin. Our arrival management problem resides in the continuous domain therefore techniques from optimal control may also be of interest. We look at the theory involved here. Although the terminology can appear quite different, there are a lot of overlaps with the ideas presented previously. There are two main approaches: Pontryagin's maximum principle [196] gives a necessary condition for a control to be optimal, approaching the continuous time problem from a calculus of variations perspective. Alternatively, we can derive the Hamilton–Jacobi–Bellman (HJB) equation via dynamic programming, which is both a necessary and sufficient condition for the optimal control. We will explore this latter approach in this appendix.

The elements of a control problem are as follows. Let there be some dynamic, continuous states of interest. We can exert controls on the system to alter these states, which in turn would affect some objective/criterion. Given the initial states, the task of a control problem is to select non-anticipative controls that would optimise our objective function. If there are uncertainties then we say that this is a stochastic (optimal) control problem. For our purposes this uncertainty is modelled as some diffusion process that can be described by Itô integrals [139].

By varying the initial times and states, we get a family of control problems. The dynamic programming method uses Bellman's Principle of Optimality to relate value functions within this family of problems and derive the HJB equation. The verification technique then finds the optimal control by taking the argmax of the Hamiltonian appearing in the HJB. In this chapter of the appendix, we describe the dynamic programming method. We start with the classical PDE approach and then state how the concept of viscosity solutions extend these ideas for more general value functions. For more in-depth discussion, see [143], [196], and the references therein. Naïve numerical methods would not ensure convergence to the viscosity solution; see methods developed for the HJB equation in e.g. [70].

B.1 The classical PDE approach

We start by making the formulation of a stochastic control problem more concrete. Let us assume that our problem lives on some given/fixed filtered probability space $(\Omega, \mathcal{F}, \{\mathcal{F}_s\}_{s\geq 0}, P)$ satisfying the usual conditions. That is, it is complete ($\mathcal{F}_0$ contains all sets of probability

zero) and right continuous

$$\mathcal{F}_{t+} = \bigcap_{s>t} \mathcal{F}_s = \mathcal{F}_t.$$

Suppose the state of the system $X_t \in \mathbb{R}^n$ follows the SDE

$$dX_s = b(X_s, a_s)dt + \sigma(X_s, a_s)dB_s, \tag{B.1}$$

where B_s is a d-dimensional Brownian motion on the given filtration. Note here we have not specified explicit dependence on the time t for the coefficients. This is important in the infinite horizon case, since we want to consider a stationary problem and find an optimal control that does not depend on the time.

Let the control space be A. The control, denoted $a_s \in A$, is a decision we make from the current available information hence it is non-anticipative – it is adapted to the filtration $\{\mathcal{F}_s\}_{s\geq 0}$. To ensure the existence and uniqueness of a strong solution to the SDE (B.1), we impose some conditions [143]. First b and σ must satisfy a Lipschitz condition: $\exists K \geq 0, \forall x, y \in \mathbb{R}^n, \forall a \in A$

$$|b(x,a) - b(y,a)| + |\sigma(x,a) - \sigma(y,a)| \leq K|x-y|. \tag{B.2}$$

Normally some continuity conditions in a, as well as some topological assumptions on A are required, see e.g. [48], [110]. From here, there are some differences between the finite and infinite horizon problems for the conditions on the control. We look at the two cases separately.

B.1.1 Finite horizon

Suppose we have a fixed time horizon with a terminal time $T < \infty$. The second condition for the existence and uniqueness of a strong solution to (B.1), in addition to (B.2), is that we only consider controls $a \in A$ such that

$$E\left[\int_0^T |b(0,a_s)|^2 + |\sigma(0,a_s)|^2 ds\right] < \infty. \tag{B.3}$$

Let us denote this set of controls which satisfy (B.3) by $\mathcal{A}$. Let the strong solution of (B.1) starting at position x at time $s = t$ be $X_s^{t,x}$.

Just as we have seen for reinforcement learning problems, we can introduce a reward for each time and a terminal reward for finite horizon problems. Let f be the running cost/reward, and g the terminal cost/reward. For comparison purposes between reinforcement learning and control theory, we refer to these as rewards to be maximised. We require that f and g are measurable functions and g is bounded below or satisfies some quadratic growth condition (see [143]). Let $\mathcal{A}^{t,x}$ be the set of controls $a \in \mathcal{A}$ which satisfy

$$E\left[\int_t^T |f(t, X_s^{t,x}, a_s)|ds\right] < \infty.$$

A control which satisfies this requirement $a \in \mathcal{A}^{t,x}$ is said to be feasible/admissible. We assume that the set of feasible controls is not empty.

The objective function for the finite horizon problem is given by

$$J(t,x,a) = E\left[\int_0^T f(t, X_s^{t,x}, a_s)ds + g(X_T^{t,x})\right], \tag{B.4}$$

$\forall (t,x) \in [0,T] \times \mathbb{R}^n$ and $a \in \mathcal{A}^{t,x}$. The value function is defined as the maximum expected reward over all feasible controls

$$v(t,x) = \sup_{a \in \mathcal{A}^{t,x}} J(t,x,a), \tag{B.5}$$

and any control a^* which achieves this value function, i.e. $J(t,x,a^*) = v(t,x)$, is known as an *optimal control*.

B.1.2 Infinite horizon

In the infinite horizon case, we denote by $\mathcal{A}$ the set of controls for which (B.3) is satisfied for all $T > 0$. Restricting the controls to this set along with (B.2) ensures that there exists a unique strong solution starting at position x at time $t = 0$. We denote this strong solution as X_s^x. As mentioned before, in order to obtain a stationary solution to the control problem in the infinite horizon case, we exclude direct dependence on time.

Let $\mathcal{A}^x$ be the controls $a \in \mathcal{A}$ which satisfy

$$E\left[\int_0^\infty e^{-\beta s} |f(X_s^x, a_s)| ds\right] < \infty.$$

Then for all $x \in \mathbb{R}^n$ and $a \in \mathcal{A}^x$ (assuming $\mathcal{A}^x$ is not empty), the objective function is given by

$$J(x,a) = E\left[\int_0^\infty e^{-\beta s} f(X_s^{t,x}, a_s) ds\right]. \tag{B.6}$$

As in the finite horizon case, the value function is defined to be

$$v(x) = \sup_{a \in \mathcal{A}^x} J(x,a), \tag{B.7}$$

and for a given x, we say that a^* is an optimal control if $J(x,a^*) = v(x)$.

The formulation presented above, where the filtered probability space is fixed, is known as the *strong* formulation. Under the *weak* formulation, the filtered probability space forms part of the control that we have to determine. The weak formulation can then be used for proofs, e.g. of the dynamic programming equation (B.8) and (B.9). For the rigorous definitions of both formulations as well as proofs on the existence of controls under these formulations, see e.g. [110] and [196].

B.1.3 Bellman's Principle of Optimality

Now that we have set up these two types of stochastic control problems, we introduce the solution method. Bellman's *Principle of Optimality* [18] states that "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." That is, we can break the overall problem down into subproblems and consider the first part of the control separately. Let θ be a stopping time. We can formulate the Principle of Optimality as follows [143].

Theorem B.1.1 (Dynamic programming principle (DPP) – Continuous case)**.**

(1) Finite horizon: let $(t,x) \in [0,T] \times \mathbb{R}^n$. *Then we have*

$$\begin{aligned} v(t,x) &= \sup_{a \in \mathcal{A}^{t,x}} \sup_{\theta \in [t,T]} E\left[\int_t^\theta f(s, X_s^{t,x}, a_s)ds + v(\theta, X_\theta^{t,x})\right] \\ &= \sup_{a \in \mathcal{A}^{t,x}} \inf_{\theta \in [t,T]} E\left[\int_t^\theta f(s, X_s^{t,x}, a_s)ds + v(\theta, X_\theta^{t,x})\right]. \end{aligned} \tag{B.8}$$

(2) Infinite horizon: let $x \in \mathbb{R}^n$. *Then we have*

$$\begin{aligned} v(x) &= \sup_{a \in \mathcal{A}^x} \sup_{\theta > 0} E\left[\int_0^\theta e^{-\beta s} f(X_s^x, a_s)ds + e^{-\beta\theta} v(X_\theta^x)\right] \\ &= \sup_{a \in \mathcal{A}^x} \inf_{\theta > 0} E\left[\int_0^\theta e^{-\beta s} f(X_s^x, a_s)ds + e^{-\beta\theta} v(X_\theta^x)\right], \end{aligned} \tag{B.9}$$

with the convention that $e^{-\beta\theta(\omega)} = 0$ *when* $\theta(\omega) = \infty$.

Proof of Theorem B.1.1 can be found in [143]. Note that the DPP can often appears as

$$v(t,x) = \sup_{a \in \mathcal{A}^{t,x}} E\left[\int_t^\theta f(s, X_s^{t,x}, a_s)ds + v(\theta, X_\theta^{t,x})\right],$$

and

$$v(x) = \sup_{a \in \mathcal{A}^x} E\left[\int_0^\theta e^{-\beta s} f(X_s^x, a_s)ds + e^{-\beta\theta} v(X_\theta^x)\right],$$

for some stopping time θ. In the infinite case, the DPP implies that for all $a \in \mathcal{A}^x$, and stopping time θ

$$v(x) \geq E\left[\int_0^\theta e^{-\beta s} f(X_s^x, a_s)ds + e^{-\beta\theta} v(X_\theta^x)\right]. \tag{B.10}$$

Also for any $\epsilon > 0$, there exists $a \in \mathcal{A}^x$ such that for all stopping time θ

$$v(x) - \epsilon < E\left[\int_0^\theta e^{-\beta s} f(X_s^x, a_s)ds + e^{-\beta\theta} v(X_\theta^x)\right].$$

Similar inequalities hold for the finite horizon case.

B.1.4 The Hamilton–Jacobi–Bellman (HJB) equation

From the DPP, we can derive the HJB equation. The HJB equation on an infinite horizon is given by

$$\beta v(x) - \sup_{a \in \mathcal{A}^x} [\mathcal{L}^a v(x) + f(x,a)] = 0, \quad \forall x \in \mathbb{R}^n, \tag{B.11}$$

where

$$\mathcal{L}^{\tilde{a}} v(x) = b(x,\tilde{a}) \cdot D_x v(x) + \frac{1}{2} tr\left(\sigma(x,\tilde{a})\sigma^T(x,\tilde{a}) D_x^2 v(x)\right). \tag{B.12}$$

This is often written as

$$\beta v(x) - H(x, D_x v(x), D_x^2 v(x)) = 0, \quad \forall x \in \mathbb{R}^n, \tag{B.13}$$

where

$$H(x,p,M) = \sup_{a \in \mathcal{A}^{t,x}} \left[b(x,a) \cdot p + \frac{1}{2} tr\big(\sigma(x,a)\sigma^T(x,a)M\big) + f(x,a)\right]. \tag{B.14}$$

We call the function H the *Hamiltonian* of the associated control problem. See Appendix C for the derivation of the HJB in the infinite horizon case.

The HJB of the finite horizon case is given by

$$-\frac{\partial v}{\partial t}(t,x) - \sup_{a\in\mathcal{A}^x}[\mathcal{L}^a v(t,x) + f(t,x,a)] = 0, \quad \forall (t,x) \in [0,T)\times\mathbb{R}^n, \tag{B.15}$$

where $\mathcal{L}^a$ is as defined in (B.12). The derivation of (B.15) can be found in [143]. For finite horizon, from (B.4), we also have a terminal condition associated with the PDE

$$v(T,x) = g(x), \quad \forall x \in \mathbb{R}^n. \tag{B.16}$$

Note that when the control space A is unbounded, the Hamiltonian can be infinite. Problems like this are said to be singular and we can derive the variational inequality associated with the control problem (see [143]).

The final step in the dynamic programming method is the verification technique/theorem. The verification theorem states that a smooth solution to the HJB equation is precisely the value function. This means we can solve the HJB equation for the value function, then the optimal control can be obtained by finding the argmax of the Hamiltonian. Finally we solve the system with the specific initial condition to get the optimal pair (x,a). See [143] or [196] for more details on the verification theorem.

B.2 Viscosity solutions

Whilst the classical PDE approach is very intuitive, it has the prohibitive condition that the value function needs to be sufficiently smooth. This is often not the case in control problems. Crandall and Lions [55] developed the theory of viscosity solutions for first order equations. Since then, the theory of viscosity solutions have been generalised to second order [54], and assuming that the function is only locally bounded [143].

Consider the class of nonlinear second-order partial differential equations given by

$$F(x, w(x), Dw(x), D^2w(x)) = 0, \quad x \in \mathcal{O}, \tag{B.17}$$

where $\mathcal{O} \subset \mathbb{R}^n$ is an open set, F is a continuous function of $\mathcal{O}\times\mathbb{R}\times\mathbb{R}^n\times S_n$ (S_n is the set of n by n symmetric matrices). We assume that F satisfies the ellipticity condition: for all $x\in\mathcal{O}, r\in\mathbb{R}, p\in\mathbb{R}^n, M_1, M_2 \in S_n$

$$M_1 - M_2 \in S_n^+ \implies F(x,r,p,M_1) \le F(x,r,p,M_2), \tag{B.18}$$

where S_n^+ is the set of symmetric, positive semidefinite, n by n matrices. We see that the HJB equation (B.13) is of the form (B.17) and satisfies the ellipticity condition (this assumes non-degeneracy of the control problem). For time-dependent problems (the finite horizon case), we can simply treat the time variable as the first element in the state variable, $x = (t, \tilde{x})$ in (B.17). Then $\mathcal{O} = [0,T)\times\mathcal{O}_{n-1}$, where $\mathcal{O}_{n-1}$ is an open subset of $\mathbb{R}^{n-1}$. In this case, we assume $F(t,x,r,q,p,M)$ satisfies the following parabolicity condition:

$$q_1 \ge q_2 \implies F(t,x,r,q_1,p,M) \le F(t,x,r,q_2,p,M). \tag{B.19}$$

We motivate viscosity solutions by first examining smooth solutions. A supersolution to (B.17) satisfies

$$F(x, w(x), Dw(x), D^2w(x)) \ge 0,$$

for the whole domain $\mathcal{O}$. A subsolution is similarly defined with $\leq$ instead.

Suppose w is a smooth supersolution of (B.17). Let φ be a smooth function on $\mathcal{O} = [0,T) \times \mathcal{O}_{n-1}$ and $(\bar{t}, \bar{x}) \in \mathcal{O}$ is a minimum point of $w - \varphi$. Since both w and φ are smooth, first derivatives of $w - \varphi$ would vanish (as long as $\bar{t} \neq 0$) and the second derivative is greater than 0. Using these derivatives and conditions (B.18) and (B.19) we get

$$\begin{aligned} &F\big(\bar{t}, \bar{x}, \varphi(\bar{t}, \bar{x}), \frac{\partial \varphi}{\partial t}(\bar{t}, \bar{x}), D_x\varphi(\bar{t}, \bar{x}), D_x^2\varphi(\bar{t}, \bar{x})\big) \\ \geq\ &F\big(\bar{t}, \bar{x}, w(\bar{t}, \bar{x}), \frac{\partial w}{\partial t}(\bar{t}, \bar{x}), D_x w(\bar{t}, \bar{x}), D_x^2 w(\bar{t}, \bar{x})\big) \geq 0. \end{aligned}$$

Since we are not assuming our functions are smooth, we require the concept of semicontinuous functions and upper and lower semicontinuous envelopes.

Definition B.2.1. *A function f is said to be upper (respectively lower) semicontinuous at a point x_0 in its domain if*

$$f(x_0) \geq \limsup_{x \to x_0} f(x) \quad \big(\textit{respectively } f(x_0) \leq \liminf_{x \to x_0} f(x)\big).$$

Definition B.2.2. *Given a locally bounded function $w : \mathcal{O} \to \mathbb{R}$. The upper (respectively lower) semicontinuous envelope is given by*

$$w^*(x) = \limsup_{x' \to x} w(x'), \quad \big(\textit{respectively } w_*(x) = \liminf_{x' \to x} w(x')\big), \ \forall x \in \mathcal{O}.$$

The upper semicontinuous envelope is the smallest upper semicontinuous function above w on $\mathcal{O}$. Similarly, the lower semicontinuous envelope is the largest lower semicontinuous function below w on $\mathcal{O}$. A function is continuous on $\mathcal{O}$ if $w(x) = w^*(x) = w_*(x)$ $\forall x \in \mathcal{O}$.

We are now ready to define the viscosity solution.

Definition B.2.3 (Viscosity solution)**.** *Let $w : \mathcal{O} \to \mathbb{R}$ be locally bounded.*

(i) w is a (discontinuous) viscosity subsolution of (B.17) on $\mathcal{O}$ if

$$F(\bar{x}, w^*(\bar{x}), D\varphi(\bar{x}), D^2\varphi(\bar{x})) \leq 0,$$

for all $\bar{x} \in \mathcal{O}$ and for all $\varphi \in C^2(\mathcal{O})$ such that $\bar{x}$ is a maximum point of $w^* - \varphi$.

(i) w is a (discontinuous) viscosity supersolution of (B.17) on $\mathcal{O}$ if

$$F(\bar{x}, w_*(\bar{x}), D\varphi(\bar{x}), D^2\varphi(\bar{x})) \geq 0,$$

for all $\bar{x} \in \mathcal{O}$ and for all $\varphi \in C^2(\mathcal{O})$ such that $\bar{x}$ is a minimum point of $w_* - \varphi$.

(iii) We say that w is a (discontinuous) viscosity solution of (B.17) on $\mathcal{O}$ if it is both a viscosity subsolution and a viscosity supersolution.

It can be shown that a locally bounded value function is a viscosity supersolution to the HJB equations, (B.13) and (B.15), provided f has quadratic growth and is continuous in the state variables for all controls a.

To show the viscosity supersolution property, we have to deal with a potentially unbounded control space. Let us define for the finite horizon case

$$\mathrm{dom}(H) = \{(t, x, p, M) \in [0,T) \times \mathbb{R}^n \times \mathbb{R}^n \times S_n : H(t, x, p, M) < \infty\}. \tag{B.20}$$

We also make the following assumption.

Assumption B.2.4. *H is continuous on int(dom(H)) and there exists some continuous function $G : [0,T) \times \mathbb{R}^n \times \mathbb{R}^n \times S_n$ such that $(t,x,p,M) \in dom(H) \iff G(t,x,p,M) \geq 0$.*

Assumption B.2.4 together with the locally bounded property is sufficient to show that the value function is a viscosity subsolution of the HJB variational ineqality given by (B.21). Equivalent assumptions can be made for the infinite horizon case to show the viscosity subsolution property of the value function for (B.22).

By utilising the viscosity supersolution and subsolution properties, it can be shown that the value function is a viscosity solution to the associated HJB variational inequality, and that this solution is unique.

Theorem B.2.5.

For some continuous function G from Assumption B.2.4, the value function v is the unique viscosity solution of the HJB variational inequality given by:

(1) Finite horizon

$$\min\Bigg(-\frac{\partial v}{\partial t}(t,x) - H(t,x,D_x v(t,x), D_x^2 v(t,x)), \\ G(t,x,D_x v(t,x), D_x^2 v(t,x))\Bigg) = 0, \quad (t,x) \in [0,T) \times \mathbb{R}^n. \tag{B.21}$$

(2) Infinite horizon:

$$\min\Big(\beta v(x) - H(x, D_x v(x), D_x^2 v(t,x)), \\ G(x, D_x v(x), D_x^2 v(x))\Big) = 0, \quad x \in \mathbb{R}^n. \tag{B.22}$$

The derivation of this theorem can be found in [196], [143]. We can apply numerical schemes to find an approximate solution to this PDE. However, naïve numerical methods applied to the HJB would not ensure convergence to the viscosity solution. For more details and examples of monotone schemes that do converge to the viscosity solution, see Appendix F and [70].

Appendix C

Derivation of the HJB equation (infinite horizon case)

Using the DPP, we derive the Hamilton–Jacobi–Bellman (HJB) equation. The HJB equation can be thought of as the infinitesimal version of the DPP. We briefly go through the derivation of the HJB for the infinite horizon case.

Let us consider $\theta = h$ and a constant control $a_t = \tilde{a} \in \mathcal{A}^x$. We start with the form expressed in (B.10):

$$v(x) \geq E\left[\int_0^h e^{-\beta s} f(X_s^x, \tilde{a})ds + e^{-\beta h} v(X_h^x)\right]. \tag{C.1}$$

The next step makes use of Itô's formula.

Theorem C.0.1 (The general Itô's Formula)**.**
Let X_t be an n-dimensional Itô process with

$$dX_t = \mu dt + \sigma dB_t.$$

Let $h(t,x) = \big(h_1(t,x), \ldots, h_p(t,x)\big)$ be a C^2 map: $[0,\infty) \times \mathbb{R}^n \to \mathbb{R}^p$. Then the process

$$Y_t = h(t, X_t)$$

is also an Itô process whose kth component is given by

$$dY_k = \frac{\partial h_k}{\partial t}(t,X)dt + \sum_i \frac{\partial h_k}{\partial x_i}(t,X)dX_i + \frac{1}{2}\sum_{i,j} \frac{\partial^2 h_k}{\partial x_i \partial x_j}(t,X)dX_i dX_j, \tag{C.2}$$

where $dB_i dB_j = \delta_{ij} dt$, and $dB_i dt = dt dt = 0$.

See [139] for the proof for one-dimensional case, [48] for general cases.

We apply Itô's formula on $v(X_h^x)$ to get

$$v(X_h^x) = v(x) + \int_0^h \mathcal{L}^{\tilde{a}} v(X_s^x)ds + \int_0^h \sigma(X_s^x, \tilde{a}) D_x v(X_s^x) dB_s, \tag{C.3}$$

where $\mathcal{L}^{\tilde{a}}$ is the generator for the diffusion (B.1) given by

$$\mathcal{L}^{\tilde{a}} v(x) = b(x,\tilde{a}) \cdot D_x v(x) + \frac{1}{2} tr\left(\sigma(x,\tilde{a})\sigma^T(x,\tilde{a}) D_x^2 v(x)\right). \tag{C.4}$$

Notice that the second integral of (C.3) is a local martingale and we can ensure its expectation vanishes by choosing stopping times such that it is uniformly bounded. When we substitute (C.3) into (C.1), we obtain

$$(1 - e^{-\beta h})v(x) \geq E\left[\int_0^h e^{-\beta s} f(X_s^x, \tilde{a})ds + e^{-\beta h}\int_0^h \mathcal{L}^{\tilde{a}} v(X_s^x)ds\right]. \tag{C.5}$$

Let us divide (C.5) by h and take the limit as $h \to 0$. Then by L'Hôpital, the left hand side becomes

$$\lim_{h\to 0} \frac{(1 - e^{-\beta h})}{h} v(x) = \lim_{h\to 0} \beta e^{-\beta h} v(x) = \beta v(x),$$

and the right hand side by mean value theorem becomes $f(x, \tilde{a}) + \mathcal{L}^{\tilde{a}} v(x)$. Upon rearranging, we then have

$$\beta v(x) - [f(x, \tilde{a}) + \mathcal{L}^{\tilde{a}} v(x)] \geq 0.$$

Since this holds true for any $a \in \mathcal{A}^x$, we obtain

$$\beta v(x) - \sup_{a\in\mathcal{A}^x} [f(x, a) + \mathcal{L}^a v(x)] \geq 0. \tag{C.6}$$

By repeating the above steps with an optimal control a^*, we find that

$$\beta v(x) - [\mathcal{L}^{a^*} v(x) + f(x, a^*)] = 0, \quad \forall x \in \mathbb{R}^n.$$

This means that equality holds in (C.6) so we obtain the HJB equation

$$\beta v(x) - \sup_{a\in\mathcal{A}^x} [\mathcal{L}^a v(x) + f(x, a)] = 0, \quad \forall x \in \mathbb{R}^n. \tag{C.7}$$

The steps involved in deriving the HJB equation for the finite horizon case is very similar to the infinite horizon case.

Appendix D

Hyperparameters

In this appendix, we list the parameters and describe the set-up used to produce the results seen in Section 3.3.

D.1 Toy model

The set-up for the toy problem was: a discretisation scale, $d = 5$, three possible actions, $a_{\min} = 1$, $a_{\max} = 3$, probability of an aircraft arriving into the system is $p = 0.4$, and minimum separation gap of 1 for aircraft of different sizes, 0 for the same size. Note that a gap of n means that there needs to be n timepoints without any flights landing to maintain separation.

The episode length was fixed at 200 timesteps. Each episode was initialised with a random state from the state space. Each configuration was chosen uniformly at random. This means that there might be too many aircraft in the system to resolve the conflicts. In order to not start with a penalty, we applied a zero filter onto the initialisation, so that only around 2 intervals will start with an aircraft.

We used a simple fully-connected neural net with three hidden layers for both the critic and the actor networks. Each layer in the actor network had 12 nodes. A ReLU activation function was used after each dense layer. The critic network was similarly structured but with 24 nodes in each layer. The replay memory had a maximum capacity of 200,000 and we used a batch size of 32.

The hyperparameters used for the DDPG algorithm can be found in Table D.1. Here τ is the update rate of the target as specified in Algorithm 3 and the last three columns are the parameters of the Ornstein–Uhlenbeck process used for exploration noise.

The parameters of the Ornstein–Uhlenbeck process was chosen to match the ones used in (earlier versions of) [114]. The other parameters were selected after hyperparameter optimisation by searching over several discrete options for each parameter. For the learning rate, this was over orders of magnitude $[0.01, 0.001, 0.0001]$, for discount factor over $[0.85, 0.9, 0.95]$, and for τ over $[0.01, 0.001]$.

Learning rate	Discount Factor	τ	θ	μ	σ
0.0001	0.95	0.001	0.15	0	0.3

Table D.1: Table of parameters

Adam [106] was used as the optimiser. We used the default initialisation of the weights from the Glorot uniform initialiser [78].

D.2 Other models using 100 intervals

For the other results, we used: a discretisation scale, $d = 100$, 10 possible actions, $a_{\min} = 1$, $a_{\max} = 10$, probability of an aircraft arriving into the system $p = 0.15$, minimum separation gap of 10 for aircraft of different sizes, 5 for the same size. Note that we have chosen p here so that the system gets sufficiently busy that the agent needs to change the sequence. Each episode has 1000 timesteps. Similar initialisation procedures to the toy model (Section D.1) were followed, so that around 15 of the intervals were occupied.

We also used simple fully-connected neural nets with three hidden layers for the critic and actor networks. Each layer had 64 nodes in the actor network, and 96 nodes in the critic network. The same hyperparameters as listed in Table D.1 were also used for these cases.

Note that whilst we have shown the results of fully connected neural networks here. Convolutional neural networks have also been tested. Using 1-dimensional convolution layers does not seem to be more effective. In most cases convergence to local minima is observed, this convergence typically take a greater number of iterations compared with the structure described above.

Appendix E

Multigrid inspired neural networks

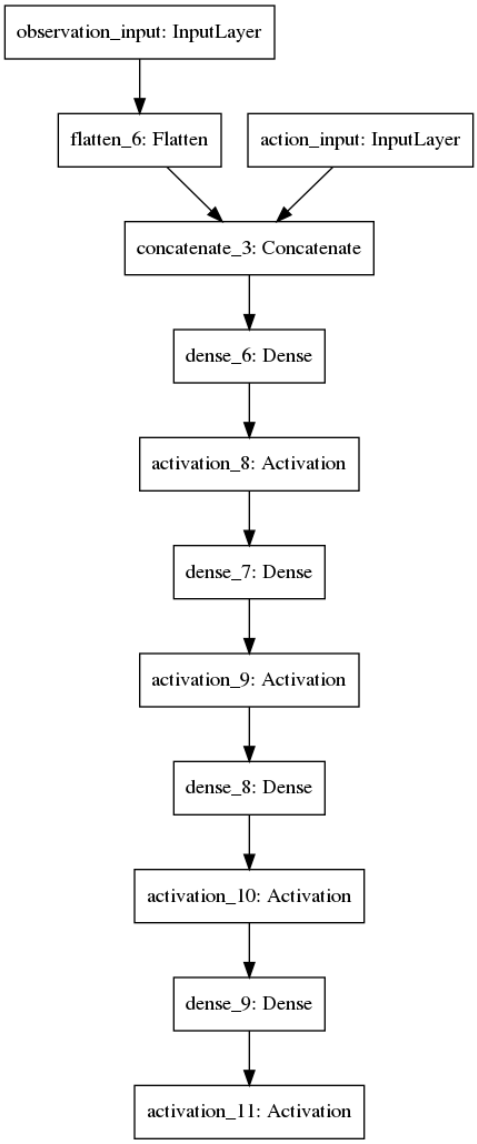


Figure E.1: Structure of the critic network.

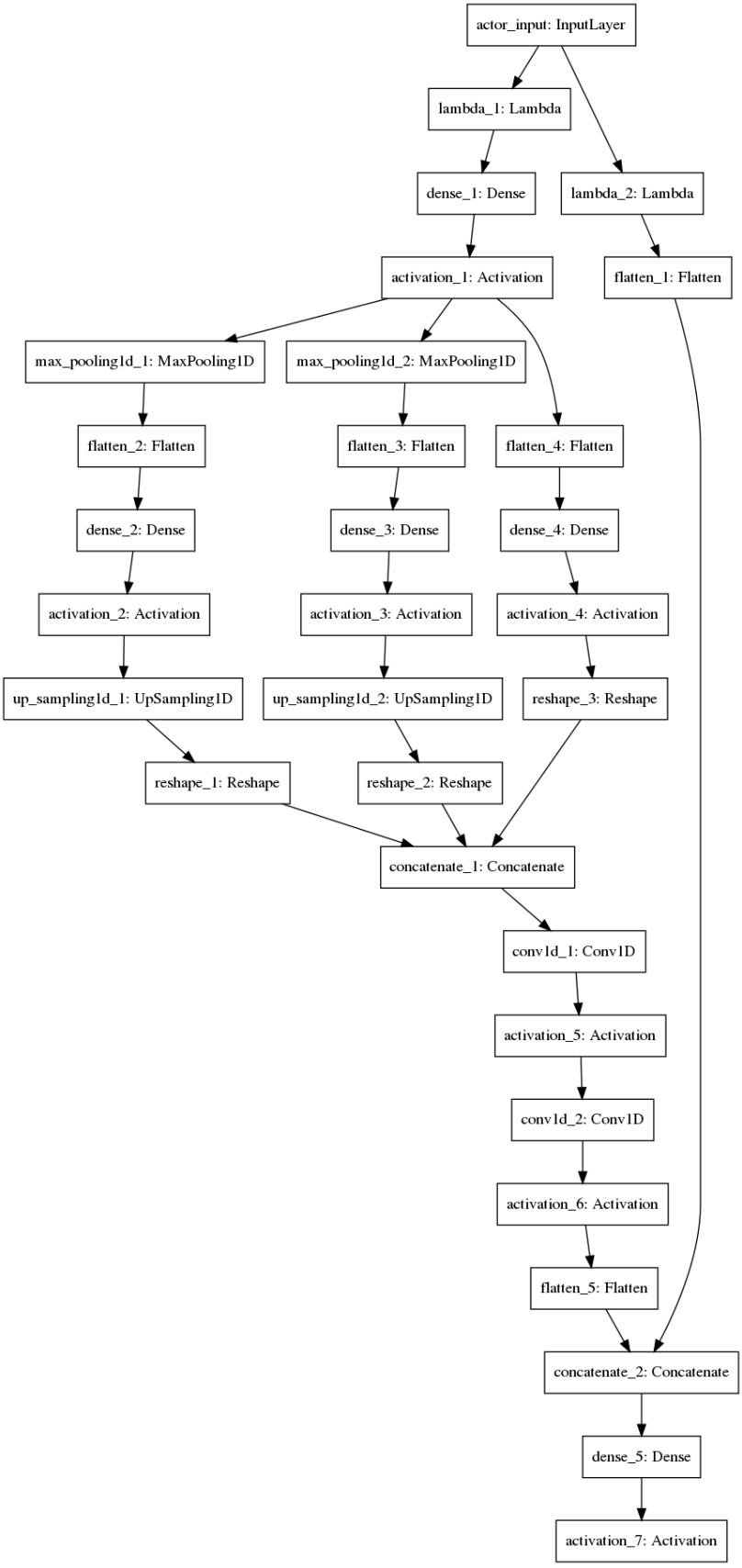


Figure E.2: Structure of the actor network.

Appendix F

Linear quadratic regulator example

To understand some of these issues involving stability and convergence of deep reinforcement learning methods, let us examine a simple linear quadratic example, where it is possible to compute the exact optimal solution. As this problem is well-known in the control literature, in this part of the appendix, we use min instead of max for the objective function, x for the state, and u for the action to align with the literature.

Let us take a linear quadratic regulator (LQR) problem. Suppose that the dynamics of the problem is linear

$$x_{t+1} = Ax_t + Bu_t,$$

and our objective function is quadratic

$$\min \sum_t x_t^\intercal Q x_t + u_t^\intercal P u_t.$$

We can design a simple experiment in `gym` and apply DDPG with a structure inspired by multi-grid methods as detailed in Section 3.3.5. For this experiment, in Figure F.1, we see that although most runs converge, there is a small fraction of examples where it is unstable. This unstable behaviour is similarly observed with a more vanilla policy gradient method by Recht [153].

We provide a possible explanation. Given our brief literature review in Section 3.4, we see monotonicity plays an importance role in policy improvement. We know that in general, monotonicity of a numerical method can have a large impact on its performance.

F.1 Monotonicity

We say a numerical method is monotone if

$$u_i^{n+1} = H(u^n, i) \quad s.t. \ \frac{\partial H(u^n, i)}{\partial u_j^n} \geq 0, \ \ \forall i, j.$$

We then have

$$v_i^n \geq u_i^n, \ \ \forall i \implies v_i^{n+1} \geq u_i^{n+1}, \ \ \forall i.$$

For linear schemes, this has the consequence that they are monotone only if all coefficients are non-negative.

Barles and Souganidis [16] proved that a stable, consistent, and monotone scheme converges (as the mesh size tends to zero) to the viscosity solution (Appendix B.2).

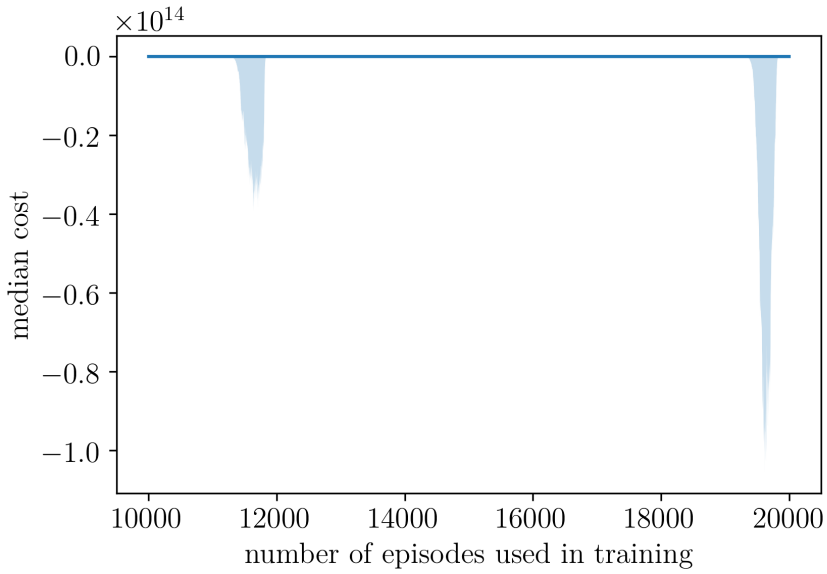


Figure F.1: Plot of the median episode cost of a LQR environment with a multigrid structure for the DDPG. The shaded region gives the range of episode costs across the runs.

We look at the impact of monotonicity in the context of linear quadratic problems, and combine this with reinforcement learning.

F.2 1D continuous LQ problem

Let us start with the continuous time problem. Consider the typical deterministic, infinite-horizon, discounted control problem with linear state dynamics

$$\frac{dX_t^{x,u}}{dt} = b(X_t^{x,u}, u_t), \quad b(X_t^{x,u}, u_t) = \alpha X_t^{x,u} + u_t, \quad X^{x,u}(0) = x, \tag{F.1}$$

where $\{X^{x,u}\}_t$ is the process starting at x and following the policy u thereafter. Let the cost function be given by

$$J(u_s; X_s) = \int_0^\infty e^{-\beta s} f(X_s^{x,u}, u_s) ds, \tag{F.2}$$

where

$$f(X_t^{x,u}, u_t) = (X_t^{x,u})^2 + u_t^2.$$

Define the value function

$$V(x) = \inf_u J(u; x). \tag{F.3}$$

Lemma F.2.1. *The dynamic programming principle gives us*

$$V(x) = \inf_u \int_0^h e^{-\beta s} \big((X_s^{x,u})^2 + u_s^2\big) ds + e^{-\beta h} V(X_h^{x,u}).$$

Proof.

$$\begin{aligned}
V(x) &= \inf_u \int_0^\infty e^{-\beta s} f(X_s^{x,u}, u_s) ds \\
&= \inf_u \int_0^h e^{-\beta s} f(X_s^{x,u}, u_s) ds + \inf_u \int_h^\infty e^{-\beta s} f(X_s^{x,u}, u_s) ds \\
&= \inf_u \int_0^h e^{-\beta s} f(X_s^{x,u}, u_s) ds + \inf_{\tilde{u}} \int_0^\infty e^{-\beta(t+h)} f(X_t^{X_h^{x,u},\tilde{u}}, \tilde{u}_t) dt \\
&= \inf_u \int_0^h e^{-\beta s} f(X_s^{x,u}, u_s) ds + e^{-\beta h} \inf_{\tilde{u}} \int_0^\infty e^{-\beta t} f(X_t^{X_h^{x,u},\tilde{u}}, \tilde{u}_t) dt \\
&= \inf_u \int_0^h e^{-\beta s} f(X_s^{x,u}, u_s) ds + e^{-\beta h} V(X_h^{x,u}).
\end{aligned}$$

■

From this we can derive the Hamilton–Jacobi–Bellman equation.

Lemma F.2.2. *The HJB equation of the system given by (F.1) and (F.2) is*

$$-\beta V(x) + \inf_u \left\{ \partial_x V(x) \cdot b(x,u) + f(x,u) \right\} = 0. \tag{F.4}$$

Proof. Let us first consider a constant control $\bar{u}$. By the definition of the value function, we must have

$$V(x) \leq \int_0^h e^{-\beta s} f(X_s^{x,\bar{u}}, \bar{u}) ds + e^{-\beta h} V(X_h^{x,\bar{u}}).$$

Now by the chain rule we have

$$V(X_h^{x,\bar{u}}) = V(x) + \int_0^h b(X_t^{x,\bar{u}}, \bar{u}) \frac{dV}{dX_t} dt,$$

therefore

$$V(x) \leq \int_0^h e^{-\beta s} f(X_s^{x,\bar{u}}, \bar{u}) ds + e^{-\beta h} \left(V(x) + \int_0^h b(X_t^{x,\bar{u}}, \bar{u}) \frac{dV}{dX_t} dt \right).$$

Upon rearranging, we have

$$\frac{1 - e^{-\beta h}}{h} V(x) \leq \frac{1}{h} \int_0^h e^{-\beta s} f(X_s^{x,\bar{u}}, \bar{u}) ds + \frac{e^{-\beta h}}{h} \int_0^h b(X_t^{x,\bar{u}}, \bar{u}) \frac{dV}{dX_t} dt.$$

We consider the limit as h tends to 0. By L'Hôpital's rule

$$\lim_{h \to 0} \frac{1 - e^{-\beta h}}{h} = \lim_{h \to 0} \frac{\beta e^{-\beta h}}{1} = \beta,$$

and by applying the Mean Value Theorem, we obtain

$$\beta V(x) \leq f(x, \bar{u}) + b(x, \bar{u}) \partial_x V(x).$$

Therefore, for an arbitrary constant cost $\bar{u}$ we have

$$-\beta V(x) + f(x, \bar{u}) + b(x, \bar{u}) \partial_x V(x) \geq 0,$$

thus

$$-\beta V(x) + \inf_u \left[f(x,u) + b(x,u)\partial_x V(x) \right] \geq 0.$$

If we apply the above analysis with the optimal control, we will find that equality holds

$$-\beta V(x) + \left[f(x,u^*) + b(x,u^*)\partial_x V(x) \right] = 0,$$

and therefore

$$-\beta V(x) + \inf_u \left[f(x,u) + b(x,u)\partial_x V(x) \right] = 0$$

as required. ■

We can find the optimal feedback control of (F.4).

Lemma F.2.3. *The optimal control is given by*

$$u^* = -\Gamma x.$$

Proof. Let us propose the following ansatz for (F.4)

$$V(x) = \Gamma x^2 + 2\kappa x + \lambda.$$

We then find the derivative with respect to x, $dV/dx = 2\Gamma x + 2\kappa$ and substitute into the Hamiltonian to get

$$V_x(x) \cdot b(x,u) + f(x,u) = Qx^2 + Ru^2 + (2\Gamma x + 2\kappa)(Ax + Bu).$$

By taking the partial derivative w.r.t u and setting to zero for the stationary point, we have that

$$2Ru + B(2\Gamma x + 2K) = 0$$

and the optimal control is given by

$$u^* = -\frac{B(\Gamma x + \kappa)}{R}.$$

Evaluate the Hamiltonian at the optimal control

$$\begin{aligned} V_x(x) \cdot b(x,u) + f(x,u)|_{u^*} &= Qx^2 + R\left(-\frac{B(\Gamma x + \kappa)}{R} \right)^2 + (2\Gamma x + 2\kappa)\left(Ax - \frac{B^2(\Gamma x + \kappa)}{R} \right) \\ &= Qx^2 + \cancel{\frac{B^2(\Gamma x + \kappa)^2}{R}} + 2A\Gamma x^2 + 2A\kappa x - \frac{2B^2(\Gamma x + \kappa)^2}{R} \\ &= \left(Q + 2A\Gamma - \frac{B^2\Gamma^2}{R} \right) + \left(2A\kappa - \frac{2B^2\Gamma\kappa}{R} \right)x - \frac{B^2\kappa^2}{R}. \end{aligned}$$

Substitute this into the HJB

$$\beta(\Gamma x^2 + \kappa x + \lambda) - \left(\left(Q + 2A\Gamma - \frac{B^2\Gamma^2}{R} \right) + \left(2A\kappa - \frac{2B^2\Gamma\kappa}{R} \right)x - \frac{B^2\kappa^2}{R} \right) = 0$$

and set each coefficient to zero

$$0 = \frac{B^2\Gamma^2}{R} + \Gamma(\beta - 2A) - Q, \tag{F.5}$$

$$0 = 2\kappa(\beta + \frac{B^2\Gamma}{R} - A), \tag{F.6}$$

$$0 = \beta\lambda + \frac{B^2\kappa^2}{R}. \tag{F.7}$$

From (F.6), we see that either $\kappa = 0$ or $\Gamma = \frac{R(A-\beta)}{B^2}$. The latter case would not generally satisfy (F.5). Thus we must have $\kappa = 0$. Substituting this into (F.7), we also get that $\lambda = 0$. Thus the only non-zero coefficient of $V(x)$ is Γ, which satisfies the quadratic (F.5). We choose the root that ensures we have a stable solution (typically positive definite).

For our problem (F.1) we have that $A = \alpha$, $B = R = Q = 1$, hence Γ must satisfy

$$\Gamma^2 + (\beta - 2\alpha)\Gamma - 1 = 0, \tag{F.8}$$

which has two roots Γ^+ and Γ^-. We choose the root that would result in a positive eigenvalue. Our optimal control is then $u^* = -\Gamma x$. ■

If we want to solve the HJB (F.4) numerically, we can rearrange the formula to get a fixed point method

$$V^{n+1} = \frac{\beta + \gamma}{\gamma} V^n + \frac{1}{\gamma} \min_u \left\{ \partial_x V^n \cdot (\alpha x + u) + x^2 + u^2 \right\}. \tag{F.9}$$

If we naïvely tried to solve this directly by approximating the derivative with finite differences, for example, central difference, then we obtain the following numerical method

$$V_i^{n+1} = \frac{\beta + \gamma}{\gamma} V_i^n + \frac{1}{\gamma} \min_u \left\{ \frac{V_{i+1}^n - V_{i-1}^n}{2\Delta x} \cdot (\alpha x_i + u) + x_i^2 + u^2 \right\}, \tag{F.10}$$

where $V_i = V(x_i)$ and $x_i = x_{i-1} + \Delta x$, and we can take forward and backward differencing at the boundary.

This method is not monotone in general as the coefficients of V_{i+1}^n and V_{i-1}^n are the opposite signs to each other.

To obtain a monotone method we look at an upwind scheme. Let us approximate the derivative w.r.t. x with forward differences or backward differences depending on the sign of $\alpha x + u$

$$V_x \approx \frac{V_{i+1} - V_i}{\Delta x} \quad \text{if} \quad \alpha x + u > 0$$

$$V_x \approx \frac{V_i - V_{i-1}}{\Delta x} \quad \text{if} \quad \alpha x + u < 0.$$

For $\alpha x + u > 0$,

$$\begin{aligned} V_i^{n+1} &= \frac{\beta + \gamma}{\gamma} V_i^n + \frac{1}{\gamma} \min_u \left\{ \frac{V_{i+1}^n - V_i^n}{\Delta x} (\alpha x_i + u) + x_i^2 + u^2 \right\} \\ &= \left(\frac{\beta + \gamma}{\gamma} - \frac{\alpha x_i + u^*}{\gamma \Delta x} \right) V_i^n + \frac{\alpha x_i + u^*}{\gamma \Delta x} V_{i+1}^n + \frac{x_i^2 + (u^*)^2}{\gamma}, \end{aligned}$$

where u^* is the argmin of the Hamiltonian. Now the coefficient of V_{i+1}^n is positive, but the coefficient of V_i^n will only be positive if we have

$$\Delta x > \frac{\alpha x_i + u^*}{\beta + \gamma}. \tag{F.11}$$

Hence taking a large value for γ enables us to use finer meshes.

Similarly when $\alpha x + u < 0$,

$$\begin{aligned} V_i^{n+1} &= \frac{\beta+\gamma}{\gamma} V_i^n + \frac{1}{\gamma} \min_u \left\{ \frac{V_i^n - V_{i-1}^n}{\Delta x}(\alpha x_i + u) + x_i^2 + u^2 \right\} \\ &= \left(\frac{\beta+\gamma}{\gamma} + \frac{\alpha x_i + u^*}{\gamma \Delta x} \right) V_i^n - \frac{\alpha x_i + u^*}{\gamma \Delta x} V_{i-1}^n + \frac{x_i^2 + (u^*)^2}{\gamma}. \end{aligned}$$

Since $\alpha x + u < 0$, the coefficient of V_{i-1}^n is positive, but the coefficient of V_i^n will only be positive if we have

$$\Delta x > -\frac{\alpha x_i + u^*}{\beta + \gamma}. \tag{F.12}$$

Note that by updating our value function as

$$V^{n+1} = \frac{\beta+\gamma}{\gamma} V^n + \frac{1}{\gamma} \min_u \{ V_x^n \cdot (ax + u) + (x^2 + u^2) \},$$

this is precisely the value iteration updates as we discussed previously. We only do one cycle of policy evaluation before choosing a new policy by taking u to be the argmin of the Hamiltonian (policy improvement). A monotone numerical scheme for the LQ problem with value iteration updates is described in Algorithm 14.

Algorithm 14: Value Iteration for LQ

1. Initialisation
Discretise the state and action spaces
Parameters: a small threshold $\theta > 0$ determining accuracy of estimation (convergence criterion), a maximum iteration count N
Initialize $V(x)$ as zeros
Initialize $\Delta > \theta$.
2. Iteration
repeat
  for *each* $x \in \mathcal{X}$ **do**
    $\hat{V}(x) \leftarrow V(x)$,
    $u^* \leftarrow$ argmin Hamiltonian at x
    $V(x) \leftarrow \frac{\beta+\gamma}{\gamma}\hat{V}(x) + \frac{1}{\gamma}H(x, u^*, \hat{V})$,
    where $H(x, u^*, \hat{V})$ is the Hamiltonian evaluated at x, with control u^* and using the suitable differencing for a monotone scheme.
  end
  $\Delta \leftarrow \max|\hat{V} - V|$.
until $\Delta < \theta$ *or* N;

For a policy iteration-like update, we need to have a fixed policy that we evaluate the value function on until convergence before we make a policy improvement step. The pseudocode is given in Algorithm 15.

Algorithm 15: Policy Iteration for LQ

1. Initialisation
Discretise the state and action spaces
Parameters: two small thresholds $\theta_v > 0$ and $\theta_u > 0$ determining accuracy of estimation (convergence criterion), maximum iteration counts N_v, N_u
Initialize $u(x) \in \mathcal{U}$ arbitrarily for all $x \in X$ (let us say equal to 1).
Initialize $\Delta > \theta$.
2. Policy Evaluation
repeat
  for *each* $x \in \mathcal{X}$ **do**
    $\hat{V}(x) \leftarrow V(x)$,
    $V(x) \leftarrow \frac{\beta+\gamma}{\gamma}\hat{V}(x) + \frac{1}{\gamma}H(x, u, \hat{V})$,
    where $H(x, u, \hat{V})$ is the Hamiltonian evaluated at x, with the current control u and using the suitable differencing for a monotone scheme.
  end
  $\Delta \leftarrow \max|\hat{V} - V|$.
until $\Delta < \theta_v$;
3. Policy Improvement
for *each* $x \in \mathcal{X}$ **do**
  $\hat{u}(x) \leftarrow u(x)$
  $u(x) \leftarrow$ argmin of Hamiltonian at x
end
If $\max|\hat{u} - u| < \theta_u$, then stop and return $V \approx V^*$ and $u \approx u^*$; else go back to Step 2.

Let us be more precise on finding the control. To recap, for the value iteration, we are updating at each iteration with the rule

$$V^{n+1} = -\frac{\beta-\gamma}{\gamma}V^n + \frac{1}{\gamma}\min_u \Big\{V_x^n \cdot (\alpha x + u) + (x^2 + u^2)\Big\}.$$

In order to obtain a monotone scheme, we must apply forward differencing on V_x^n if $\alpha x + u > 0$ or backward differencing otherwise. However, for value iteration, we are also minimising over all u, which means that there are a few cases we can fall into. Let us consider finding the correct action for each discretised state x_i.

We have the regions $R_1 = \{u : \alpha x_i + u \geq 0\}$ and $R_1 = \{u : \alpha x_i + u < 0\}$. Let

$$H^n(u) = \begin{cases} h_1^n(u) & \text{if } u \in R_1 \\ h_2^n(u) & \text{if } u \in R_2 \end{cases}$$

where

$$h_1^n = \frac{V_{i+1}^n - V_i^n}{\Delta x}(\alpha x_i + u) + x_i^2 + u^2,$$

and

$$h_2^n = \frac{V_i^n - V_{i-1}^n}{\Delta x}(\alpha x_i + u) + x_i^2 + u^2.$$

Our problem is now to find

$$y = \min_u H^n(u).$$

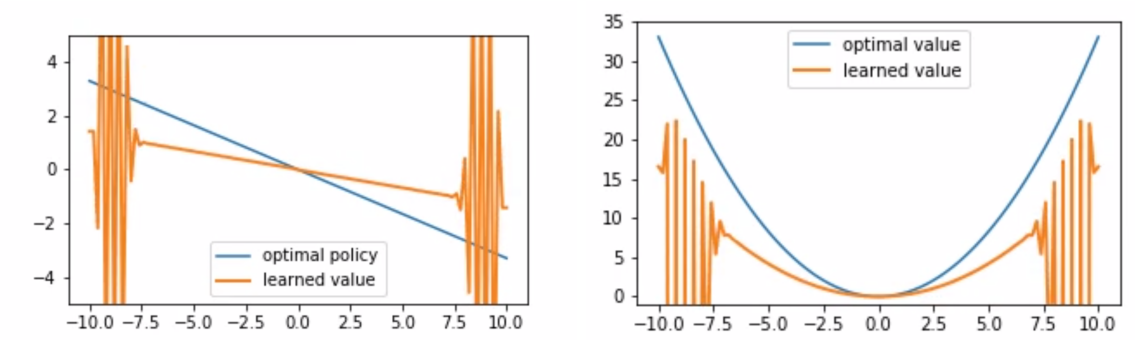


Figure F.2: An intemediate policy and value function for a downwind method. Here the policy and value function have not converged yet. The instability forms and becomes amplified with further iterations.

The smoothness of $H(u)$ depends on the smoothness of the value function. In the case of the LQ problem, the value function is smooth, therefore $H(u)$ is smooth except on the boundary of R_1 and R_2, which in this case is a linear boundary, $u = -\alpha x_i$.

We can solve the above problem numerically, by finding

$$y_1^* = \min_u h_1(u), \quad u \in R_1$$

and

$$y_2^* = \min_u h_2(u), \quad u \in R_2$$

then taking the minimum over these 2 values

$$y = \min\{y_1, y_2\}$$

to get the desired control for value iteration.

For policy iteration, when we are doing policy evaluation, as the policy is fixed, we just need to check the sign of $\alpha x + u$. However, we need to also ensure that monotonicity is maintained in policy improvement. The policy improvement step can be described as

$$u^* = \arg\min_u H(u).$$

In this case we can again solve the 2 minimisation problem separately to find y_1^* and y_2^* and choose u based on which of these have the lower value.

In Figures F.2 and F.3, we see the result of applying the downwind iteration and upwind iteration respectively. We see clearly the instability arising in the downwind case, and how important it is to choose between forward and backward difference so that we ensure monotonicity.

F.3 1D discrete LQ problem

We now turn our attention to the deterministic discrete case. Let

$$x_{t+1} = b(x_t, u_t)$$

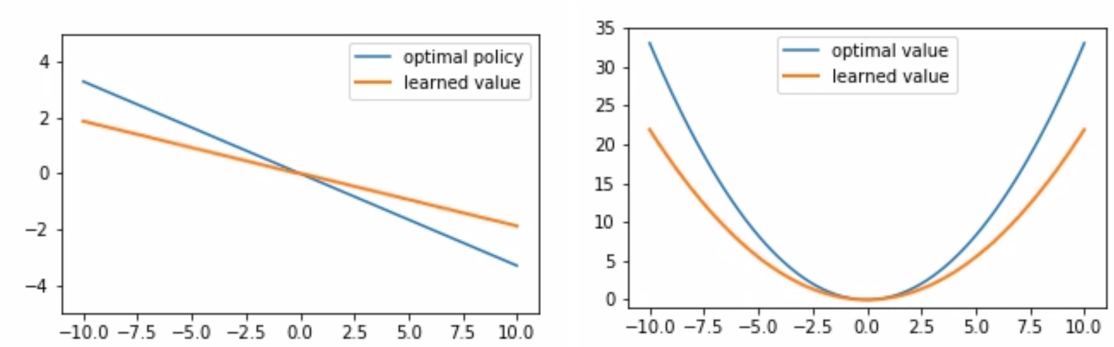


Figure F.3: An intermediate policy and value function for an upwind method. Note that whilst the policy has not converged yet, there are no instabilities in this case.

and the reward function be given by $f(x, a)$. The state value function under policy π is defined

$$V^{\pi}(x) = \sum_{t=0}^{\infty} \gamma^t f(x_t, \pi(x_t)), \quad x_0 = x.$$

The Bellman optimality equations are

$$V^*(x_t) = \max_{u_t} \Big[f(x_t, u_t) + \gamma V^*(x_{t+1})|_{x_{t+1}=b(x_t,u_t)} \Big],$$

$$\begin{aligned} Q^*(x_t, u_t) &= f(x_t, u_t) + \gamma V^*(x_{t+1})|_{x_{t+1}=b(x_t,u_t)} \\ &= f(x_t, u_t) + \gamma \min_{\hat{a}} Q(x_{t+1}, \hat{a})|_{x_{t+1}=b(x_t,u_t)}. \end{aligned}$$

Thus the first order condition is given by

$$\frac{\partial Q^*(s_{t+1}, \hat{a})}{\partial \hat{a}} = 0$$

or

$$\frac{\partial f}{\partial u_t} + \gamma \frac{\partial V^*}{\partial x_{t+1}} \frac{\partial x_{t+1}}{\partial u_t} = 0.$$

F.3.1 Q-Learning

For Q-learning, the updates to the action-value functions (Q-functions) are given by

$$\begin{aligned} Q^{n+1}(x_t, u_t) =& Q^n(x_t, u_t) + \alpha \Big[r(x_t, u_t) + \gamma \min_{\hat{u}} Q^n(x_{t+1}, \hat{u}) - Q^n(x_t, u_t) \Big] \\ =& (1 - \alpha) Q^n(x_t, u_t) + \alpha \Big[r(x_t, u_t) + \gamma \min_{\hat{u}} Q^n(x_{t+1}, \hat{u}) \Big]. \end{aligned}$$

We see the coefficients are non-negative for $0 \leq \alpha \leq 1$ (and this is usually the quoted range for α in Q-learning). This indicates that the update step is monotone.

In our numerical experiments, Q-learning was shown to be stable against the theoretical limits for a deterministic LQ problem when $0 \leq \alpha \leq 1$, see Figure F.4. Here we converge to the correct value function, and the differences in policy are due to discretisation error. Since monotonicity is a sufficient condition for convergence, having α outside of this range

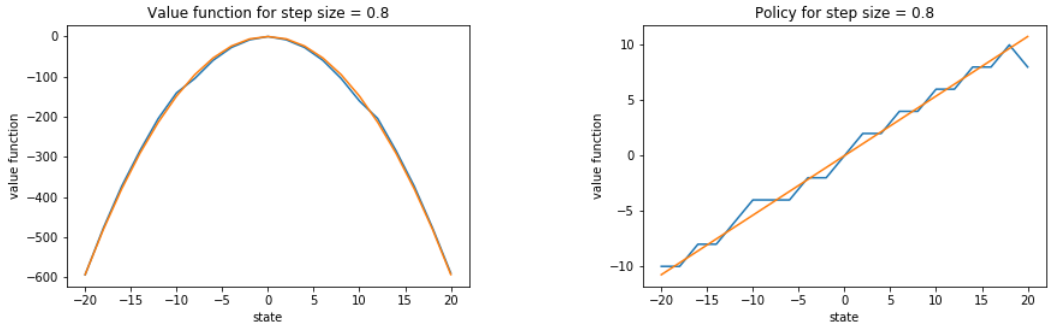


Figure F.4: Learnt value function and policy (blue) against theoretical (orange) for $\alpha = 0.8$

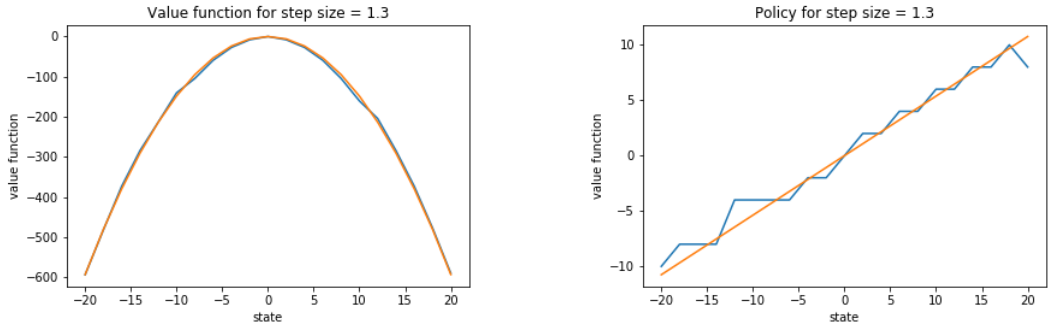


Figure F.5: Learnt value function and policy (blue) against theoretical (orange) for $\alpha = 1.3$

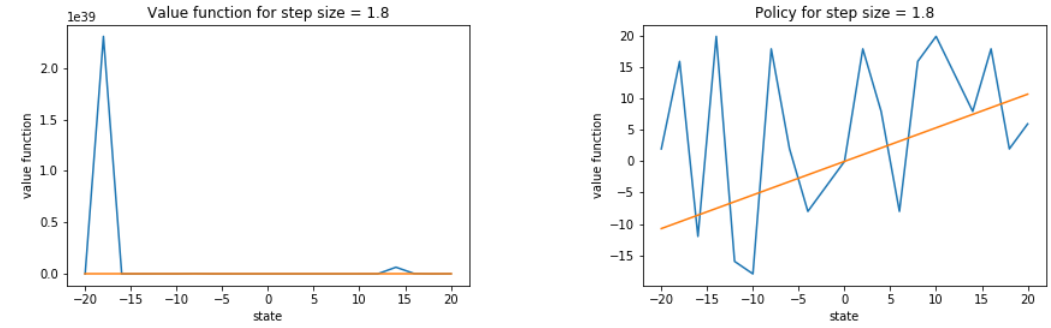


Figure F.6: Learnt value function and policy (blue) against theoretical (orange) for $\alpha = 1.8$

does not necessitate the method breaking. In Figure F.5 we see that we can still converge to the theoretical values for $\alpha = 1.3$. However when α is sufficiently large, $\alpha = 1.8$ will do, we see that we no longer converge to the true solution (Figure F.6).

From the above, it appears that having a monotone scheme would result in convergence. The problem with these table-based methods is that they are slow to converge, and the number of states/state-action pairs suffer from the curse of dimensionality. Using a function approximator, we postulate that as long as the function approximation preserves monotonicity, then Q-learning should still converge.

Note that, when we have a constant step size $\alpha > 1$, not only can we no longer guarantee monotonicity, but the square summability condition of (3.2) would also be violated.

For the 'table-lookup' methods, as long as all states are updated infinitely often and step sizes satisfy (3.2), then we have convergence for policy evaluation. This is not guaranteed when we have a general function approximator. When it comes to function approximation, there are no guarantees whether the approximating set will even contain the optimal value function. The key questions are:

1. Does the given algorithm converge?
2. If so, is the limit close to the optimal value function?
3. If not, is it within a fixed distance away from the optimal value function?

When we are approximating $Q(x,u)$ by $\tilde{Q}(x,u,w)$, we want to minimise the expected error

$$\frac{1}{2}E\big[(Q(x,u)-\tilde{Q}(x,u,w))^2\big].$$

We use a semi-gradient descent method and take update steps in the form of

$$w_{n+1}=w_n+\alpha_n\big(Q(x,u)-\tilde{Q}(x,u,w_n)\big)\nabla_{w_n}\tilde{Q}(x,u,w_n).$$

For reinforcement learning/control problems, we do not have access to the true value function $Q(x,u)$ and therefore use an approximation in its place. An approximate method derived from Q-learning uses

$$Q(x,u)\approx r(x,u)+\gamma\max_{\bar{u}}\tilde{Q}(x',\bar{u},w).$$

Since

$$Q^*(x_t,u)=f(x_t,u)+V^*(x_{t+1})$$

and we know in the LQ case, both f and V can be expressed as a quadratic function in x and u, we can use a linear function approximator going up to quadratic terms.

Suppose we have a naïve linear approximator for $Q(x,u)$

$$\tilde{Q}(x,u,w)=X(x,u)^\intercal w, \tag{F.13}$$

where $X(x,u)$ are the features that we extract from our state-action pair, and w are the weights. Then

$$\begin{aligned}\tilde{Q}^{n+1}(x,u,w_{n+1}) &= X^\intercal(x,u)\big(w_n+\alpha_n\big(Q(x,u)-\tilde{Q}^n(x,u,w_n)\big)X(x,u)\big)\\ &\approx \tilde{Q}^n(x,u,w_n)\\ &\quad+\alpha_n X^\intercal(x,u)\big(r(x,u)+\gamma\max_{\tilde{u}}\tilde{Q}^n(x',\tilde{u},w_n)-\tilde{Q}^n(x,u,w_n)\big)X(x,u)\big)\\ &= \alpha_n r(x,u)X^\intercal(x,u)X(x,u)+(1-\alpha_n X^\intercal(x,u)X(x,u))\tilde{Q}^n(x,u,w_n)\\ &\quad+\alpha_n X^\intercal(x,u)X(x,u)\max_{\tilde{u}}\tilde{Q}^n(x',\tilde{u},w_n).\end{aligned} \tag{F.14}$$

Monotonicity implies that the action-value function $\tilde{Q}^{n+1}(x,u,w_{n+1})$ must be non-decreasing with respect to the action-value function at the other points. Hence to ensure monotonicity we need to ensure that the features $X(x,u)$ are bounded and we take sufficiently small step size such that $\alpha_n X^\intercal(x,u)X(x,u)<1$, i.e. $\alpha_n(x,u)<1/(X^\intercal(x,u)X(x,u))$, the step-size is dependent on the state and action. A potential problem is that we may not sufficiently explore the state space with a step size that is dependent on the state-action space.

F.3.2 DPG

Q-learning seems to be well-behaved, but value approximation algorithms are known to be unstable in policy (particularly if the value function is not very smooth). For this reason, methods for directly approximating the policy gained popularity $\pi_\theta(x,u) \approx \pi(x,u)$. Actor-critic methods approximate both value function and policy, since an approximate for the value function can reduce the variance.

The DPG [165] is an actor-critic method and updates the parameters by

$$\delta_t = r_t + \gamma \tilde{Q}(x_{t+1}, \pi_{\theta_t}(x_{t+1}), w_t) - \tilde{Q}(x_t, u_t, w_t) \tag{F.15}$$

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w \tilde{Q}(x_t, u_t, w_t) \tag{F.16}$$

$$\theta_{t+1} = \theta_t + \alpha_\theta \nabla_\theta \pi_{\theta_t}(x_t) \nabla_u \tilde{Q}(x_t, u_t, w_t)|_{u_t = \pi_{\theta_t}(x_t)}. \tag{F.17}$$

Note that if we have an on-policy algorithm, then $u_t = \pi_\theta(x_t)$, otherwise for off-policy algorithms, u_t is arbitrary.

Suppose we have a linear function approximator as in (F.13). In this case, we have

$$\nabla_w Q^w(x,u) = X(x,u).$$

Take any reference point $(x_t, u_t) = (\tilde{s}, \tilde{a})$. Monotonicity implies that the value function $Q^{n+1}(\tilde{s}, \tilde{a})$ must be non-decreasing with respect to the value function at the other points $Q^n(p,q)$.

As we have a linear function approximator in Q,

$$\begin{aligned}
Q^{n+1}(x_t,u_t) =& w^{n+1} X(x_t,u_t) \\
=& (w^n + \alpha_w \delta_t \nabla_w Q^w(x_t,u_t)) X(x_t,u_t) \\
=& \Big(w^n + \alpha_w \big(r_t + \gamma Q^w(x_{t+1}, \pi_{\theta_t}(x_{t+1})) - Q^w(x_t,u_t) \big) \nabla_w Q^w(x_t,u_t) \Big) X(x_t,u_t) \\
=& \big(w^n + \alpha_w r_t X^T(x_t,u_t) \big) X(x_t,u_t) + \alpha_w \gamma X^T(x_t,u_t) X(x_t,u_t) Q^w(x_{t+1}, \pi_{\theta_t}(x_{t+1})) \\
& - \alpha_w X^T(x_t,u_t) X(x_t,u_t) Q^w(x_t,u_t).
\end{aligned}$$

The coefficient for $Q^w(x_{t+1}, \pi_{\theta_t}(x_{t+1}))$ is positive and the coefficient for $Q^w(x_t,u_t)$ is negative. Therefore this method is not necessarily monotone. This can be part of the issues relating to convergence of the DDPG algorithm, which utilises the deterministic policy gradient.

Appendix G

Proof of Proposition 8.1.3

Proposition G.0.1. *Suppose that* $k < b_2 - b_1$ *and either*

$$\frac{a_2 - a_1 - k}{b_2 - b_1 - k} < \gamma < \min\left(\frac{b_2 - b_1 - k}{a_2 - a_1 + k}, \frac{a_2 - a_1 + k}{b_2 - b_1 - k}\right)$$

or

$$\max\left(\frac{b_2 - b_1 - k}{a_2 - a_1 + k}, \frac{a_2 - a_1 + k}{b_2 - b_1 - k}\right) < \gamma < \frac{b_2 - b_1 + k}{a_2 - a_1 + k},$$

alternatively

$$k < \frac{(b_2 - b_1) - (a_2 - a_1)}{2}$$

and

$$\frac{(a_2 - a_1) + k}{(b_2 - b_1) - k} < \gamma < \frac{(b_2 - b_1) - k}{(a_2 - a_1) + k},$$

then there is no basin of attraction of the $(p, q) = (1, 0)$ *corner, and all trajectories converge to* $(p, q) = (0, 1)$.

Recall that for the anti-coordination setting we should have either $a_1 < a_2$ and $b_1 < b_2$, or $a_1 > a_2$ and $b_1 > b_2$ (that is, the agents prefer the same action). We assume without loss of generality that $a_1 < a_2$ and $b_1 < b_2$ (so that action 1 is preferable for both agent) and also that $b_2 - b_1 > a_2 - a_1$. For convenience, we recap some key equations here. The amended partial derivatives with considerate costs are

$$\frac{\partial \tilde{l}_A}{\partial p} = \dot{p} = -2\gamma kp + 2kq - (a_2 - a_1) + \gamma(b_2 - b_1) + (\gamma - 1)k,$$

$$\frac{\partial \tilde{l}_B}{\partial q} = \dot{q} = 2kp - 2\gamma kq + \gamma(a_2 - a_1) - (b_2 - b_1) + (\gamma - 1)k,$$

so

$$\frac{dq}{dp} = \frac{2kp - 2\gamma kq + \gamma(a_2 - a_1) - (b_2 - b_1) + (\gamma - 1)k}{-2\gamma kp + 2kq - (a_2 - a_1) + \gamma(b_2 - b_1) + (\gamma - 1)k}.$$

When do we get stuck at the undesirable corner of $(1, 0)$? This happens when near $(1, 0)$, the descent direction of the agent is in the positive x, negative y direction. That is, when

$\dot{p} < 0$ and $\dot{q} > 0$. Now suppose if $b_2 - b_1 > k$, then

$$\begin{aligned}\dot{p}|_{(1,0)} &= -2\gamma k - (a_2 - a_1) + \gamma(b_2 - b_1) + (\gamma - 1)k < 0 \\ &\iff \gamma((b_2 - b_1) - k) < (a_2 - a_1) + k \\ &\implies \gamma < \frac{(a_2 - a_1) + k}{(b_2 - b_1) - k}\end{aligned}$$

$$\begin{aligned}\dot{q}|_{(1,0)} &= 2k + \gamma(a_2 - a_1) - (b_2 - b_1) + (\gamma - 1)k > 0 \\ &\iff \gamma > \frac{(b_2 - b_1) - k}{(a_2 - a_1) + k}. \end{aligned} \tag{G.1}$$

So if $b_2 - b_1 > k$, then gradient steps take us towards the undesirable corner when

$$\frac{(b_2 - b_1) - k}{(a_2 - a_1) + k} < \gamma < \frac{(a_2 - a_1) + k}{(b_2 - b_1) - k}.$$

This has feasible γ values when

$$\begin{aligned}((b_2 - b_1) - k)^2 &< ((a_2 - a_1) + k)^2 \\ (b_2 - b_1)^2 - 2k(b_2 - b_1) &< (a_2 - a_1)^2 + 2k(a_2 - a_1) \\ (b_2 - b_1)^2 - (a_2 - a_1)^2 &< 2k((a_2 - a_1) + (b_2 - b_1)) \\ k &> \frac{(b_2 - b_1) - (a_2 - a_1)}{2}.\end{aligned}$$

In the case where $b_2 - b_1 < k$, the condition for γ from $\dot{p}$ becomes

$$\gamma > \frac{(a_2 - a_1) + k}{(b_2 - b_1) - k}.$$

But we note that the right hand side is negative, and therefore any $\gamma \geq 0$ satisfies this. Similarly the right hand side of (G.1) is also negative, and therefore again $\gamma \geq 0$ satisfies this automatically. Hence if $b_2 - b_1 < k$, there are no additional constraints on γ; there is a basin of attraction for all $\gamma > 0$.

Finally, in the case where $b_2 - b_1 = k$, we again find that there are no additional restrictions imposed, and there is a basin of attraction for the $(1, 0)$ corner (since $\dot{p} < 0, \dot{q} > 0$ for $\gamma > 0$.

Thus if we have one of

1. $b_2 - b_1 \leq k$,
2. Both
$$\frac{(b_2 - b_1) - (a_2 - a_1)}{2} < k < b_2 - b_1$$
and
$$\frac{(b_2 - b_1) - k}{(a_2 - a_1) + k} < \gamma < \frac{(a_2 - a_1) + k}{(b_2 - b_1) - k},$$

then near to $(1,0)$ we would move towards it and get stuck.

In the case where $\dot{p} > 0$ and $\dot{q} < 0$, we are moving towards the $(0,1)$ corner as desired. We check the conditions for this. Reversing the signs above, we get

$$\dot{p}|_{(1,0)} > 0 \Rightarrow \gamma > \frac{(a_2 - a_1) + k}{(b_2 - b_1) - k} \quad \text{if } b_2 - b_1 > k,$$

$$\dot{q}|_{(1,0)} < 0 \Rightarrow \gamma < \frac{(b_2 - b_1) - k}{(a_2 - a_1) + k}. \tag{G.2}$$

We can check when there are feasible solutions to γ, i.e. when

$$\frac{(a_2 - a_1) + k}{(b_2 - b_1) - k} < \frac{(b_2 - b_1) - k}{(a_2 - a_1) + k}.$$

Rearranging, we find

$$\begin{aligned}
((a_2 - a_1) + k)^2 &< ((b_2 - b_1) - k)^2 \\
(a_2 - a_1)^2 + 2k(a_2 - a_1) &< (b_2 - b_1)^2 - 2k(b_2 - b_1) \\
2k((a_2 - a_1) + (b_2 - b_1)) &< (b_2 - b_1)^2 - (a_2 - a_1)^2 \\
k &< \frac{(b_2 - b_1) - (a_2 - a_1)}{2}.
\end{aligned}$$

Note that the above is a tighter upper bound on k than $b_2 - b_1$.

Thus we move towards to correct corner when

$$k < \frac{(b_2 - b_1) - (a_2 - a_1)}{2}$$

and

$$\frac{(a_2 - a_1) + k}{(b_2 - b_1) - k} < \gamma < \frac{(b_2 - b_1) - k}{(a_2 - a_1) + k}.$$

Given the assumption that $\gamma > 0$, the condition on $\dot{q}$ (G.2) cannot be satisfied if $b_2 - b_1 < k$. Likewise, the case $b_2 - b_1 = k$ is the same as described above, any $\gamma > 0$ will give convergence to the undesirable corner.

Note that when $\frac{dq}{dp} > 0$, we can move towards either $(1,1)$ or $(0,0)$ corner. Therefore we need to check that the gradients locally near these corners are such that the agents move from these corners too, which is what we examine next.

Aside from the case analysed above, $\frac{dq}{dp} > 0$ near (1,0) would also avoid converging to $(p,q) = (1,0)$. Now

$$\begin{aligned}
\left.\frac{dq}{dp}\right|_{(1,0)} &= \frac{2k + \gamma(a_2 - a_1) - (b_2 - b_1) + (\gamma - 1)k}{-2\gamma k - (a_2 - a_1) + \gamma(b_2 - b_1) + (\gamma - 1)k} \\
&= -1 + \frac{(\gamma - 1)\big((a_2 - a_1) + (b_2 - b_1)\big)}{-(a_2 - a_1) + \gamma(b_2 - b_1) - (\gamma + 1)k} \\
&= -1 + \frac{(1 - \gamma)\big((a_2 - a_1) + (b_2 - b_1)\big)}{(a_2 - a_1) - \gamma(b_2 - b_1) + (\gamma + 1)k}.
\end{aligned}$$

Hence

$$\left.\frac{dq}{dp}\right|_{(1,0)} > 0 \iff \frac{(1-\gamma)\big((a_2-a_1)+(b_2-b_1)\big)}{(a_2-a_1)-\gamma(b_2-b_1)+(\gamma+1)k} > 1$$
$$\implies \frac{(1-\gamma)}{\big((a_2-a_1)+k\big)-\gamma\big((b_2-b_1)-k\big)} > \frac{1}{(a_2-a_1)+(b_2-b_1)}.$$

For notational convenience, let us set (note that in fact $\beta = -\dot{p}$)

$$\alpha = \frac{1-\gamma}{\big((a_2-a_1)+k\big)-\gamma\big((b_2-b_1)-k\big)}, \tag{G.3}$$
$$\beta = \big((a_2-a_1)+k\big)-\gamma\big((b_2-b_1)-k\big),$$
$$\mu = \frac{1}{(a_2-a_1)+(b_2-b_1)}.$$

Recall that in order to avoid having a basin of attraction, we want $\alpha > \mu$. Let us split this into the following cases:

(i) $\beta > 0$,

(ii) $\beta < 0$,

(iii) $\beta = 0$

Case (i): $\beta > 0$

We have $\alpha > \mu$ when

$$\begin{aligned}(1-\gamma)\big((b_2-b_1)+(a_2-a_1)\big) &> (a_2-a_1)+k-\gamma\big((b_2-b_1)-k\big)\\ (b_2-b_1)-\gamma(a_2-a_1) &> k+\gamma k\\ \gamma &< \frac{(b_2-b_1)-k}{(a_2-a_1)+k} \quad \text{note denominator is positive by assump.}\end{aligned}$$

Case (ii): $\beta < 0$

We reverse the inequality above, so $\alpha > \mu$ when

$$\gamma > \frac{(b_2-b_1)-k}{(a_2-a_1)+k}.$$

Case (iii): $\beta = 0$

Here, we have $k+(a_2-a_1) = ((b_2-b_1)-k)\gamma$, so that

$$\gamma = \frac{(a_2-a_1)+k}{(b_2-b_1)-k},$$

since $\beta = -\dot{p}|_{(1,0)} = 0$, this implies that there is a p-nullcline on $q=0$. To escape the corner $(1,0)$, we require $\dot{q} < 0$, that is

$$\begin{aligned}2k-(b_2-b_1)+\gamma(a_2-a_1)+(\gamma-1)k &< 0\\ -\big((b_2-b_1)-k\big)+\gamma\big((a_2-a_1)+k\big) &< 0\\ \frac{-\big((b_2-b_1)-k\big)^2+\big((a_2-a_1)+k\big)^2}{(b_2-b_1)-k} &< 0.\end{aligned}$$

If the denominator is positive ($k < b_2 - b_1$), then we require that the numerator is negative

$$-\big((b_2 - b_1) - k\big)^2 + \big((a_2 - a_1) + k\big)^2 < 0$$
$$(a_2 - a_1)^2 + 2k(a_2 - a_1) - (b_2 - b_1)^2 + 2k(b_2 - b_1) < 0$$
$$k < \frac{(b_2 - b_1)^2 - (a_2 - a_1)^2}{2(b_2 - b_1) + (a_2 - a_1)} = \frac{(b_2 - b_1) - (a_2 - a_1)}{2},$$

which is a tighter upper-bound on k than $b_2 - b_1$. Alternatively, if the denominator is negative ($k > b_2 - b_1$), then the numerator must be positive in that range as $2k(b_2 - b_1) > (b_2 - b_1)^2$.

Therefore, in Case (iii), we move away from (1,0) when $k > b_2 - b_1$ or

$$k < \frac{(b_2 - b_1) - (a_2 - a_1)}{2}.$$

Note that this does not contradict the derivations earlier when we found the ranges at which we get stuck at (1,0).

Conditions on k for Cases (i) and (ii)

Let us focus on the conditions for k under Cases (i) and (ii) by splitting the values of k into three possibilities:

1. $k > b_2 - b_1$,
2. $k = b_2 - b_1$,
3. $k < b_2 - b_1$.

Case 1: $k > b_2 - b_1$

From the analysis earlier, we noted that in this case, for all values of γ there is a basin of attraction. We show an alternative derivation below.

For Case (i), $\beta > 0$, we need

$$\beta = \big((a_2 - a_1) + k\big) - \gamma\big((b_2 - b_1) - k\big) > 0$$
$$\gamma > \frac{(a_2 - a_1) + k}{(b_2 - b_1) - k},$$

as the right hand side is negative, this is trivially satisfied for all $\gamma \geq 0$. However, recall from the Case (i) analysis above that $\alpha > \mu$ if

$$\gamma < \frac{(b_2 - b_1) - k}{(a_2 - a_1) + k} < 0,$$

which cannot be satisfied.

For Case (ii), $\beta < 0$, we have

$$\beta = \big((a_2 - a_1) + k\big) - \gamma\big((b_2 - b_1) - k\big) < 0$$
$$\gamma < \frac{(a_2 - a_1) + k}{(b_2 - b_1) - k} < 0,$$

which is not possible as we require $\gamma \geq 0$.

Case 2: $k = b_2 - b_1$

Note that we have

$$\begin{aligned}\alpha =& \frac{1-\gamma}{(b_2-b_1)+(a_2-a_1)}\\ =& \mu - \frac{\gamma}{(b_2-b_1)+(a_2-a_1)} < \mu,\end{aligned}$$

but this contradicts the requirement that $\alpha > \mu$. Therefore this value of k is infeasible.

Case 3: $k < b_2 - b_1$

For Case (i), $\beta > 0$, so that

$$\begin{aligned}\big((a_2-a_1)+k\big) - \gamma\big((b_2-b_1)-k\big) &> 0\\ \gamma &< \frac{(a_2-a_1)+k}{(b_2-b_1)-k},\end{aligned}$$

and $\alpha > \mu$ in this case if

$$\gamma < \frac{(b_2-b_1)-k}{(a_2-a_1)+k}.$$

Therefore we require

$$\gamma < \min\left(\frac{(b_2-b_1)-k}{(a_2-a_1)+k}, \frac{(a_2-a_1)+k}{(b_2-b_1)-k}\right).$$

Likewise for Case (ii), $\beta < 0$, we need

$$\gamma > \max\left(\frac{(b_2-b_1)-k}{(a_2-a_1)+k}, \frac{(a_2-a_1)+k}{(b_2-b_1)-k}\right).$$

Direction of flow for Case 3

To find the direction of the flow, recall that

$$\begin{aligned}\dot{p}|_{(1,0)} =& -2\gamma k - (a_2-a_1) + \gamma(b_2-b_1) + (\gamma-1)k\\ =& \gamma((b_2-b_1)-k) - ((a_2-a_1)+k).\end{aligned}$$

This means that for Case (i) where

$$\gamma < \frac{(a_2-a_1)+k}{(b_2-b_1)-k},$$

we have $\dot{p}|_{(1,0)} < 0$ and so the flow is in the direction of the $(1,1)$ corner. For Case (ii) where

$$\gamma > \frac{(a_2-a_1)+k}{(b_2-b_1)-k},$$

we have $\dot{p}|_{(1,0)} > 0$, and so the flow is in the direction of the $(0,0)$ corner.

We note that the flow will keep going in the positive q direction or the negative p near the two boundaries due to the forms of $\dot{p}$ and $\dot{q}$. Thus we need to also determine the behaviour at the top right and bottom left corner and ensure the flow is towards the $(p,q) = (0,1)$ corner.

Flow at (1,1)

Now the flow at (1,1) should be in the negative p direction, so we need a positive gradient for p (recall the flow is in the descent direction). Since

$$\begin{aligned}\dot{p}|_{(1,1)} &= -2\gamma k + 2k - (a_2 - a_1) + \gamma(b_2 - b_1) + (\gamma - 1)k \\ &= \gamma((b_2 - b_1) - k) - ((a_2 - a_1) - k),\end{aligned}$$

$$\begin{aligned}\dot{p}|_{(1,1)} > 0 &\iff \gamma((b_2 - b_1) - k) - ((a_2 - a_1) - k) > 0 \\ &\implies \gamma > \frac{(a_2 - a_1) - k}{(b_2 - b_1) - k}.\end{aligned}$$

Thus in the case where from $(1, 0)$, the agents move towards $(0, 1)$ via $(1, 1)$, we need

$$\frac{(a_2 - a_1) - k}{(b_2 - b_1) - k} < \gamma < \min\left(\frac{(b_2 - b_1) - k}{(a_2 - a_1) + k}, \frac{(a_2 - a_1) + k}{(b_2 - b_1) - k}\right). \tag{G.4}$$

We can also check values of γ exist in this range. Now clearly

$$\frac{(a_2 - a_1) - k}{(b_2 - b_1) - k} < \frac{(a_2 - a_1) + k}{(b_2 - b_1) - k}.$$

For feasible values of γ where

$$\frac{(a_2 - a_1) - k}{(b_2 - b_1) - k} < \frac{(b_2 - b_1) - k}{(a_2 - a_1) + k},$$

we need

$$\begin{aligned}((a_2 - a_1) - k)((a_2 - a_1) + k) &< ((b_2 - b_1) - k)^2 \quad \text{note denoms are positive} \\ (a_2 - a_1)^2 - k^2 &< (b_2 - b_1)^2 - 2k(b_2 - b_1) + k^2 \\ 2k^2 - 2(b_2 - b_1)k + (b_2 - b_1)^2 - (a_2 - a_1)^2 &> 0.\end{aligned}$$

This quadratic inequality in k has solution

$$k < \frac{(b_2 - b_1) - \sqrt{2(a_2 - a_1)^2 - (b_2 - b_1)^2}}{2} =: k^-, \tag{G.5}$$

$$k > \frac{(b_2 - b_1) + \sqrt{2(a_2 - a_1)^2 - (b_2 - b_1)^2}}{2} =: k^+. \tag{G.6}$$

So when do we need to ensure (G.5) and (G.6)? We look at the case when

$$\begin{aligned}\frac{(b_2 - b_1) - k}{(a_2 - a_1) + k} &< \frac{(a_2 - a_1) + k}{(b_2 - b_1) - k} \\ ((b_2 - b_1) - k)^2 &< ((a_2 - a_1) + k)^2 \\ (b_2 - b_1)^2 - 2k(b_2 - b_1) &< (a_2 - a_1)^2 + 2k(a_2 - a_1) \\ (b_2 - b_1)^2 - (a_2 - a_1)^2 &< 2k((b_2 - b_1) + (a_2 - a_1)).\end{aligned}$$

$$\begin{aligned}\implies k &> \frac{(b_2 - b_1) + (a_2 - a_1))((b_2 - b_1) - (a_2 - a_1))}{2((b_2 - b_1) + (a_2 - a_1))} \\ k &> \frac{(b_2 - b_1) - (a_2 - a_1)}{2},\end{aligned}$$

so we know that

$$\frac{(b_2 - b_1) - (a_2 - a_1)}{2} < k < b_2 - b_1. \tag{G.7}$$

However (G.5) and (G.6) gives tighter bounds on k, so we can refine the range of k to be

$$\frac{(b_2 - b_1) - (a_2 - a_1)}{2} < k < k^-$$
$$k^+ < k < b_2 - b_1$$

if there are 2 distinct roots to the quadratic. If the discriminant is less than 0, then any values of k will give feasible values of γ for (G.4), this occurs when $(b_2 - b_1) > \sqrt{2}(a_2 - a_1)$.

To summarise, if

$$k < \frac{(b_2 - b_1) - (a_2 - a_1)}{2}, \tag{G.8}$$

then the minimum of the two terms in (G.4) means that the inequality is automatically satisfied.

If $(b_2 - b_1) > \sqrt{2}(a_2 - a_1)$, then the inequality (G.4) holds for all k.

Else we need

$$\frac{(b_2 - b_1) - (a_2 - a_1)}{2} < k < k^- \tag{G.9}$$
$$k^+ < k < b_2 - b_1. \tag{G.10}$$

Flow to (0,0)

We need to move upwards and so we want to have a negative gradient for q, hence we need

$$\begin{aligned} & \dot{q}|_{(0,0)} < 0 \\ \iff & \gamma(a_2 - a_1) - (b_2 - b_1) + (\gamma - 1)k < 0 \\ \iff & \gamma\big((a_2 - a_1) + k\big) - \big((b_2 - b_1) + k\big) < 0 \\ \iff & \gamma < \frac{(b_2 - b_1) + k}{(a_2 - a_1) + k}. \end{aligned}$$

Therefore for the trajectory of moving from near the $(1, 0)$ corner to the $(0, 1)$ corner via $(0, 0)$ corner, we need

$$\max\left(\frac{(b_2 - b_1) - k}{(a_2 - a_1) + k}, \frac{(a_2 - a_1) + k}{(b_2 - b_1) - k}\right) < \gamma < \frac{(b_2 - b_1) + k}{(a_2 - a_1) + k}. \tag{G.11}$$

In order to have feasible values of γ, we can check that the upper bound is greater than the lower bound. Clearly we have

$$\frac{(b_2 - b_1) - k}{(a_2 - a_1) + k} < \frac{(b_2 - b_1) + k}{(a_2 - a_1) + k},$$

so when does

$$\frac{(a_2 - a_1) + k}{(b_2 - b_1) - k} < \frac{(b_2 - b_1) + k}{(a_2 - a_1) + k}?$$

Noting that the denominator is positive, we need

$$\begin{aligned}((a_2-a_1)+k)^2 <&((b_2-b_1)+k)((b_2-b_1)-k)\\ (a_2-a_1)^2+2k(a_2-a_1)+k^2 <&(b_2-b_1)^2-k^2\\ 2k^2+2k(a_2-a_1)-((b_2-b_1)^2-(a_2-a_1)^2) <&0.\end{aligned}$$

By finding the roots of the quadratic, we get that

$$\frac{-(a_2-a_1)-\sqrt{2(b_2-b_1)^2-(a_2-a_1)^2}}{2}<k<\frac{-(a_2-a_1)+\sqrt{2(b_2-b_1)^2-(a_2-a_1)^2}}{2}, \tag{G.12}$$

note that the discriminant is strictly positive, so there will always be two roots.

Finally we check when

$$\frac{(b_2-b_1)-k}{(a_2-a_1)+k}<\frac{(a_2-a_1)+k}{(b_2-b_1)-k}.$$

But note that we have already checked this above, the condition on k is given by (G.7).

Combining with (G.12), we get the restrict range of

$$\frac{(b_2-b_1)-(a_2-a_1)}{2}<k<\frac{-(a_2-a_1)+\sqrt{2(b_2-b_1)^2-(a_2-a_1)^2}}{2}. \tag{G.13}$$

So to summarise the case for this corner, either the above inequality (G.13) holds, or if

$$k<\frac{(b_2-b_1)-(a_2-a_1)}{2},$$

then there are guaranteed to be feasible values of γ satisfying (G.11) to ensure convergence to the $(0,1)$ corner.

Therefore we arrive at the conditions that if $k<b_2-b_1$ and either

$$\frac{(a_2-a_1)-k}{(b_2-b_1)-k}<\gamma<\min\left(\frac{(b_2-b_1)-k}{(a_2-a_1)+k},\frac{(a_2-a_1)+k}{(b_2-b_1)-k}\right)$$

or

$$\max\left(\frac{(b_2-b_1)-k}{(a_2-a_1)+k},\frac{(a_2-a_1)+k}{(b_2-b_1)-k}\right)<\gamma<\frac{(b_2-b_1)+k}{(a_2-a_1)+k},$$

then there is the desired convergence behaviour. We have further refined the range of k to ensure there are feasible values of γ in this range. This is precisely given by (a combination of) (G.8), (G.9), (G.10), and (G.13) depending on the flow.

Appendix H

Proof of Proposition 8.2.1

As to ensure convergence to the correct solution in a subgame with one overlapping action, we perform similar analysis as before. We suppose that agent B has more to gain, therefore $b_m - b_i > a_j - a_i$.

As a reminder, we have

$$\frac{dq}{dp} = \frac{kp - \gamma kq - (b_m - b_i) + \gamma(a_j - a_i)}{-\gamma kp + kq - (a_j - a_i) + \gamma(b_m - b_i)}.$$

To reach $(0, 1)$, we look at the behaviour at $(1, 0)$. As discussed previously in Appendix G, when $\frac{dq}{dp}\big|_{(1,0)} > 0$, we move away from this corner. For $\frac{dq}{dp} > 0$, either

(a) both $\dot{p} > 0$ and $\dot{q} > 0$, or

(b) both $\dot{p} < 0$ and $\dot{q} < 0$

Assume: $b_m - b_i > k$

Suppose for now that $b_m - b_i > k$. For $\dot{q}|_{(1,0)} > 0$, we have $k + b_i - b_m - \gamma(a_i - a_j) > 0$ so $\gamma > \frac{(b_m - b_i) - k}{a_j - a_i}$. For $\dot{p}|_{(1,0)} > 0$, we have $-\gamma k + a_i - a_j - \gamma(b_i - b_m) > 0$ so $\gamma > \frac{a_j - a_i}{(b_m - b_i) - k}$.

Therefore, for Case (a) (both $\dot{p} > 0$ and $\dot{q} > 0$), we need

$$\gamma > \max\left(\frac{(b_m - b_i) - k}{a_j - a_i}, \frac{a_j - a_i}{(b_m - b_i) - k}\right).$$

For Case (a), as $\dot{p} > 0$, the flow is towards $(0, 0)$.

Similarly, for Case (b) ($\dot{p} < 0$ and $\dot{q} < 0$), we need

$$\gamma < \min\left(\frac{(b_m - b_i) - k}{a_j - a_i}, \frac{a_j - a_i}{(b_m - b_i) - k}\right).$$

For Case (b), as $\dot{p} < 0$, the flow is towards $(1, 1)$.

Flow at (1,1)

At $(1, 1)$, we want $\dot{p} > 0$ so that we flow in the negative p direction. Now $\dot{p}|_{(1,1)} > 0$

$$\begin{aligned}
&\iff -\gamma k + k + (a_i - a_j) - \gamma(b_i - b_m) > 0 \\
&\iff -(a_j - a_i - k) + \gamma(b_m - b_i - k) > 0 \\
&\iff \gamma > \frac{a_j - a_i - k}{b_m - b_i - k}
\end{aligned}$$

Hence we need

$$\frac{(a_j - a_i) - k}{(b_m - b_i) - k} < \gamma < \min\left(\frac{(b_m - b_i) - k}{a_j - a_i}, \frac{a_j - a_i}{(b_m - b_i) - k}\right).$$

Clearly

$$\frac{(a_j - a_i) - k}{(b_m - b_i) - k} < \frac{a_j - a_i}{(b_m - b_i) - k}.$$

For

$$\frac{(a_j - a_i) - k}{(b_m - b_i) - k} < \frac{(b_m - b_i) - k}{a_j - a_i}, \tag{H.1}$$

we need

$$(a_j - a_i)^2 - k(a_j - a_i) < (b_m - b_i)^2 - 2k(b_m - b_i) + k^2$$
$$k^2 + k(-2(b_m - b_i) + (a_j - a_i)) + (b_m - b_i)^2 - (a_j - a_i)^2 > 0.$$

The roots of this quadratic are

$$k = \frac{2(b_m - b_i) - (a_j - a_i) \pm \sqrt{5(a_j - a_i)^2 - 4(b_m - b_i)(a_j - a_i)}}{2}.$$

Let us call the roots k^- and k^+. When $b_m - b_i > 5/4(a_j - a_i)$, the discriminant is negative. That is, there are no real roots to the quadratic, which means the inequality holds for all k. Otherwise, the inequality will be satisfied when $k < k^-$ and $k > k^+$.

When is (H.1) the one that we are interested in? That is, when is

$$\frac{(b_m - b_i) - k}{a_j - a_i} < \frac{a_j - a_i}{(b_m - b_i) - k}?$$

This occurs when $(b_m - b_i)^2 - 2k(b_m - b_i) + k^2 - (a_j - a_i)^2 < 0$. The roots are

$$\begin{aligned} k &= \frac{\cancel{2}(b_m - b_i) \pm \sqrt{\cancel{4}(b_m - b_i)^2 - \cancel{4}((b_m - b_i)^2 - (a_j - a_i)^2)}}{\cancel{2}} \\ &= (b_m - b_i) \pm (a_j - a_i). \end{aligned}$$

That is, the inequality holds when $(b_m - b_i) - (a_j - a_i) < k < (b_m - b_i) + (a_j - a_i)$, which by our assumption refines to

$$(b_m - b_i) - (a_j - a_i) < k < (b_m - b_i). \tag{H.2}$$

To summarise we need

$$\frac{(a_j - a_i) - k}{(b_m - b_i) - k} < \gamma < \min\left(\frac{(b_m - b_i) - k}{a_j - a_i}, \frac{a_j - a_i}{(b_m - b_i) - k}\right).$$

When $b_m - b_i > 5/4(a_j - a_i)$, we have feasible solution for γ, for all $0 < k < b_m - b_i$. Otherwise, in order to have feasible γ, we need either $k < (b_m - b_i) - (a_j - a_i)$ or $k^+ < k < b_m - b_i$ or $(b_m - b_i) - (a_j - a_i) < k < k^-$.

Flow at (0,0)

At $(0,0)$, we need $\dot{q} < 0$, so

$$kp - \gamma kq + b_i - b_m - \gamma(a_i - a_j)|_{(0,0)} < 0$$
$$\gamma < \frac{b_m - b_i}{a_j - a_i}.$$

Hence

$$\max\left(\frac{(b_m - b_i) - k}{a_j - a_i}, \frac{a_j - a_i}{(b_m - b_i) - k}\right) < \gamma < \frac{b_m - b_i}{a_j - a_i}.$$

Now

$$\frac{(b_m - b_i) - k}{a_j - a_i} < \frac{b_m - b_i}{a_j - a_i},$$

for any $k \geq 0$. We have

$$\frac{a_j - a_i}{(b_m - b_i) - k} < \frac{b_m - b_i}{a_j - a_i} \tag{H.3}$$

if

$$(a_j - a_i)^2 < (b_m - b_i)^2 - k(b_m - b_i)$$
$$k < \frac{(b_m - b_i)^2 - (a_j - a_i)^2}{(b_m - b_i)}. \tag{H.4}$$

It remains to check when the above inequality (H.3) is the one of interest, i.e.

$$\frac{a_j - a_i}{(b_m - b_i) - k} > \frac{(b_m - b_i) - k}{a_j - a_i}.$$

But note this has already been analysed above, the condition obtained is (H.2) $(b_m - b_i) - (a_j - a_i) < k < (b_m - b_i) + (a_j - a_i)$.

Notice

$$\frac{(b_m - b_i)^2 - (a_j - a_i)^2}{(b_m - b_i)} < \frac{((b_m - b_i) - (a_j - a_i))((b_m - b_i) + (a_j - a_i))}{(b_m - b_i) - (a_j - a_i)}$$
$$= (b_m - b_i) + (a_j - a_i).$$

Hence we refine the bounds (H.2) with (H.4) to

$$(b_m - b_i) - (a_j - a_i) < k < \frac{(b_m - b_i)^2 - (a_j - a_i)^2}{(b_m - b_i)}.$$

In the above analysis, we assumed that $b_m - b_i > k$, now let us examine the other cases.

Assume: $b_m - b_i < k$

In order to have $\dot{p} > 0$ at $(1,0)$, we need

$$\gamma < \frac{a_j - a_i}{(b_m - b_i) - k} < 0,$$

which contradicts $\gamma > 0$. In order to have $\dot{q} < 0$, we would require

$$\gamma < \frac{(b_m - b_i) - k}{a_j - a_i} < 0,$$

which is again not possible by assumption on γ at $(1,0)$. Hence we cannot have $\frac{dq}{dp} > 0$ when $b_m - b_i < k$.

Assume: $b_m - b_i = k$

Now if $b_m - b_i = k$, then $\dot{p} > 0$ implies that $a_i - a_j > 0$, which contradicts the assumption that $a_i < a_j$. And $\dot{q} < 0$ implies that $-\gamma(a_i - a_j) < 0$ which is again not possible. Therefore in this case we also cannot have $\frac{dq}{dp} > 0$.

Suppose $\frac{dq}{dp} < 0$

Finally we look at what happens when $\frac{dq}{dp} < 0$. First, note that we get stuck if $\dot{p} < 0$ and $\dot{q} > 0$ at $(1, 0)$. We discussed above when $k > b_m - b_i$ or $k = b_m - b_i$, contradictions occur when $\dot{p} > 0$ or $\dot{q} < 0$, hence we must have $\dot{p} < 0$ and $\dot{q} > 0$. Hence we get stuck at (1,0).

Now let us assume $k < b_m - b_i$. Then $\dot{p} < 0$ and $\dot{q} > 0$ at $(1, 0)$ when

$$-\gamma k - (a_j - a_i) + \gamma(b_m - b_i) < 0$$
$$\gamma < \frac{a_j - a_i}{(b_m - b_i) - k},$$

and

$$k - (b_m - b_i) + \gamma(a_j - a_i) > 0$$
$$\gamma > \frac{(b_m - b_i) - k}{a_j - a_i}.$$

We have previously derived that

$$\frac{(b_m - b_i) - k}{a_j - a_i} < \frac{a_j - a_i}{(b_m - b_i) - k}$$

precisely when k satisfies

$$(b_m - b_i) - (a_j - a_i) < k < (b_m - b_i) + (a_j - a_i).$$

So if k satisfies the above inequality and

$$\frac{(b_m - b_i) - k}{a_j - a_i} < \gamma < \frac{a_j - a_i}{(b_m - b_i) - k},$$

then there is a basin of attraction for the social sub-optimum.

We move towards $(0, 1)$ if $\dot{p} > 0$ and $\dot{q} < 0$, which happens when

$$\gamma > \frac{a_j - a_i}{(b_m - b_i) - k}$$

and

$$\gamma < \frac{(b_m - b_i) - k}{a_j - a_i}.$$

Now we need

$$\frac{(b_m - b_i) - k}{a_j - a_i} > \frac{a_j - a_i}{(b_m - b_i) - k},$$

which will happen when

$$0 < k < (b_m - b_i) - (a_j - a_i)$$

or

$$k > (b_m - b_i) + (a_j - a_i).$$

But we note the latter is not possible as we also require $k < b_m - b_i$.

Appendix I

Numerical results for multi-agent anti-coordination games

In Chapter 8, we have derived conditions for the two-player, two-action game for the socially optimum convergence behaviour (and the case with one overlapping action), as well as designed algorithms to scale this up to higher numbers of actions and players, we look at the performance of the proposed methods in this appendix.

I.1 Two-actions

First let us take a look at a two-action example where players optimising their own objectives do not converge to the social optimum.

Suppose that we have costs for each agent as seen in Table I.1. Using the derived formulas in Section 8.1, optimising the welfare function with these specific costs, gives us the system

$$\begin{pmatrix}\dot{p}\\ \dot{q}\end{pmatrix} = \begin{pmatrix}0 & 4k\\ 4k & 0\end{pmatrix}\begin{pmatrix}p\\ q\end{pmatrix} - \begin{pmatrix}2k+1\\ 2k+2\end{pmatrix}.$$

When we use each individual's loss function for the gradient descent, then we obtain the system

$$\begin{pmatrix}\dot{p}\\ \dot{q}\end{pmatrix} = \begin{pmatrix}0 & 2k\\ 2k & 0\end{pmatrix}\begin{pmatrix}p\\ q\end{pmatrix} - \begin{pmatrix}k+1\\ k+2\end{pmatrix}.$$

Finally, by (8.7), the system with considerate cost is given by

$$\begin{pmatrix}\dot{p}\\ \dot{q}\end{pmatrix} = \begin{pmatrix}-2\gamma k & 2k\\ 2k & -2\gamma k\end{pmatrix}\begin{pmatrix}p\\ q\end{pmatrix} + \begin{pmatrix}k(\gamma-1)+2\gamma-1\\ k(\gamma-1)+\gamma-2\end{pmatrix}.$$

The case when $k = 1$ is interesting as there is a dominant strategy for each agent. The pure Nash equilibrium is for each player to choose action 1, even though this is not a Pareto optimal strategy since $(p, q) = (0, 1)$ results in only a cost of $(1, 1)$ compared with $(1, 2)$ when both players choose action 1. For any $k > 0$, we note that $(p, q) = (0, 1)$ is always Pareto optimal, and for $k > 0.5$, $(p, q) = (0, 1)$ is the social optimum. If the conflict cost is set for example to $k = 1$, we end up with a saddle-like surface for the social welfare as seen in Figure I.1.

		Agent B	
		1	2
Agent A	1	$(k, 1+k)$	$(0, 3)$
	2	$(1, 1)$	$(1+k, 3+k)$

Table I.1: Costs of actions for a two-action game.

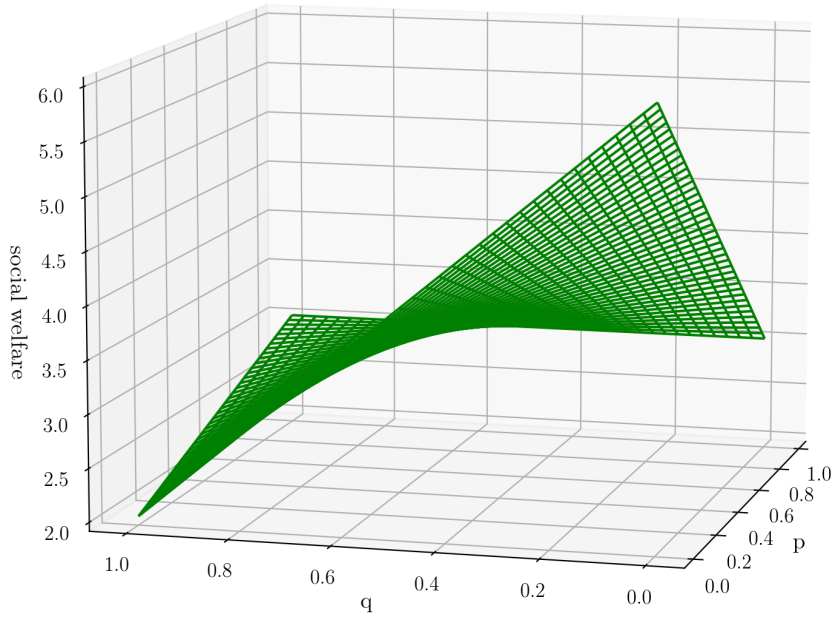


Figure I.1: 3D surface plot of the social objective for $k = 1$.

When optimising the welfare function, the stationary point (which is a saddle point) occurs at $(p, q) = \left(\frac{k+1}{2k}, \frac{2k+1}{4k}\right)$. The eigenvalues of the matrix in the autonomous system are $\pm 4k$ and the eigenvectors are given by $(1, 1)^\intercal$ and $(1, -1)^\intercal$.

If each agent optimises their own loss function for the gradient descent, the interior stationary point occurs at $(p, q) = \left(\frac{k+2}{2k}, \frac{k+1}{2k}\right)$, the eigenvalues are now $\pm 2k$ and the eigenvectors remain the same as the welfare function case.

We see that regardless of whether we are optimising by the welfare function, L, or the individual functions l_i, the interior equilibrium point converges to $(1/2, 1/2)$ as k increases. A simple phase portrait for the welfare function case is as illustrated in Figure I.2.

As mentioned earlier, if $k = 1$, our system has a pure Nash equilibrium at $(p, q) = (1, 1)$ which is not the social optimum. If we optimise on an individual basis, then we converge to the Nash equilibrium. We can infer this behaviour from the 3D plot of the social welfare function in Figure I.1. The surface at $p = 1$ lies very flat (the figure has been rotated so that this is on the far side of the diagram) whereas the other side has both the maximum and the minimum value. We see that the minimum/social optimum is $(p, q) = (0, 1)$.

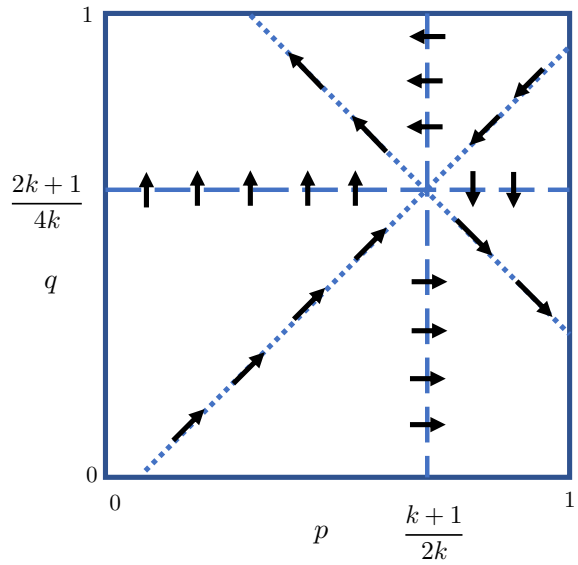


Figure I.2: A simplified phase portrait for optimising the welfare function. We have marked on the nullclines and the eigenvectors, as well as the descent directions.

To illustrate the impact that the considerate cost has, for the $k = 1$ case, the quiver plots and simulated trajectories for optimising the social welfare, or each individual functions, along with the addition of considerate costs at different weights can be seen in Figure I.3. In this figure, it can be observed that both the individual and welfare objective gets stuck along the boundary as the boundaries are on the nullspace here. Introducing $\gamma = 0.4$ for the considerate cost resolves this issue. In the lower right subplot, we can clearly see the nullspace formed by a zero eigenvalue when $\gamma = 1$.

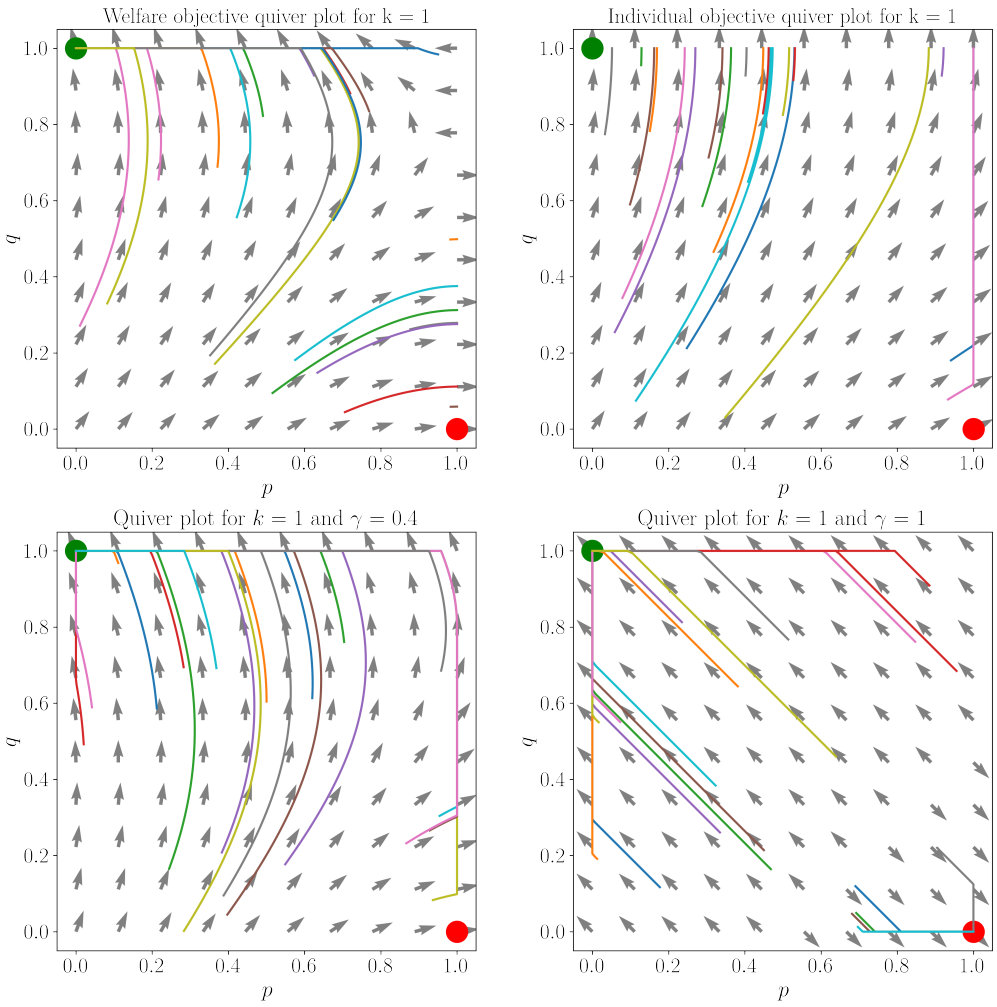


Figure I.3: Quiver plots and simulated trajectories for optimising the welfare function (top left), the individual objectives(top right) and with considerate costs for $\gamma = 0.4$ (bottom left), and $\gamma = 1.0$ (bottom right) for $k = 1$. The green dot represents the social optimum (when $k > 0.5$), and the red dot represents the other Pareto optimum (when k is sufficiently large these are the only pure strategy Pareto optimum).

The direction of the vector field is such that for $\gamma \leq 0.5$, we converge to $(p, q) = (0, 1)$ for all starting positions (seen in the lower left plot of Figure I.3). With γ increasing to 1, the line $q = p - 1/2$ forms and bisects the region into two parts. For 7/8 of the region, we converge to $(p, q) = (0, 1)$, that is the pure strategy which corresponds to the social optimum. It is interesting to note that this is an increased region compared with optimising the welfare function (which converges to $(p, q) = (0, 1)$ for only 23/32 of initial points).

Once we have enforced the constraints specified in Proposition 8.1.3, all trajectories converge to the desired corner as can be seen in Figure I.4.

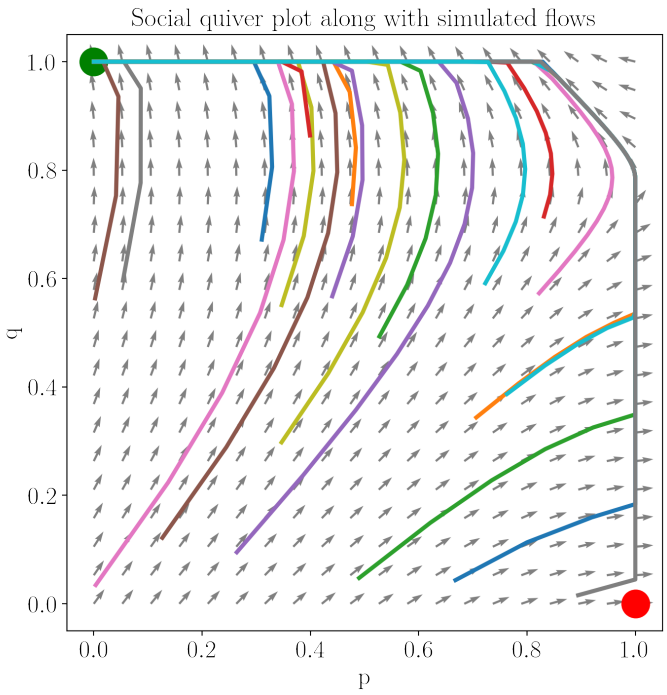
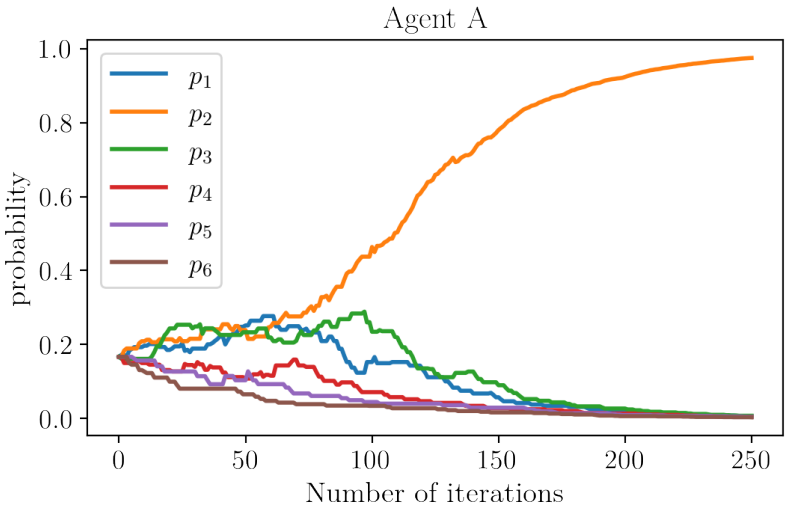


Figure I.4: Example trajectories after we enforce the constraints.

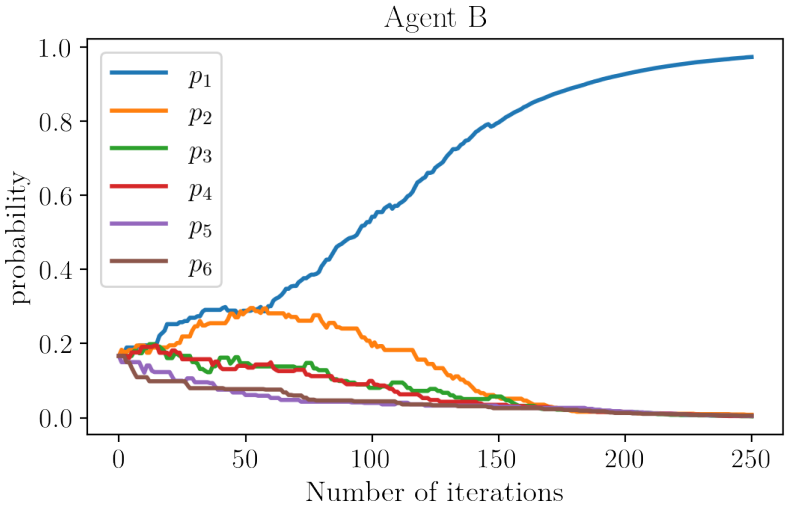
I.2 Many actions

We tackle problems involving more than 2 actions by applying Algorithm 8. Let the vectors of preferences for the agents A and B be $a = [0, 1.2, 2.1, 2.9, 3.4, 6.1]$ and $b = [1.4, 3.1, 4.2, 6.1, 6.3, 7]$. Suppose we start each agent with equal probability of taking any of the actions, we run Algorithm 8 with $w = 0.1$ to evolve these probabilities. At each stage, we ensure that the relevant conditions of the subgame (either Proposition 8.1.3 or Proposition 8.2.1) are satisfied.

As agent B has more to gain by taking the first action compared with agent A, the social optimum is $p_A = [0, 1, 0, 0, 0, 0]$, $p_B = [1, 0, 0, 0, 0, 0]$. We see from Figure I.5a and Figure I.5b that this is indeed what the algorithm converges to.



(a) Agent A's evolving probabilities.



(b) Agent B's evolving probabilities.

Figure I.5: Evolving probabilities for a two-player, many-actions example.

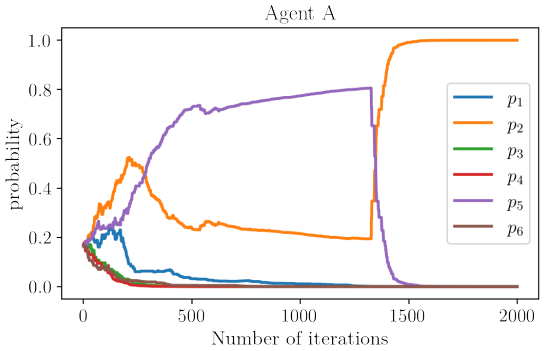
I.3 Many agents

We demonstrate that Algorithm 9 successfully finds the social optimum on an example of 4 agents with 6 actions.

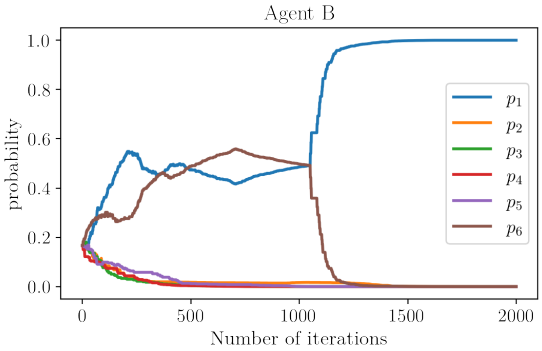
Let the preferences of the agents be $a = [0, 12, 21, 29, 34, 61]$, $b = [14, 31, 42, 61, 63, 70]$, $c = [7, 18, 22, 41, 83, 90]$, and finally $d = [10, 20, 30, 31, 44, 99]$. To achieve social optimum, we expect that agent A would choose action 2, agent B would choose action 1, agent C would choose action 3, and finally agent D would choose action 4. Figure I.6a to Figure I.6d gives the results, and we see each agent selects the actions as expected. During training we observe that agents A and B wait for the others to confirm their choices before choosing their best action, due to the considerate cost imposed.

Finally in Figure I.7, we see what would happen if we modified Algorithm 9 so that

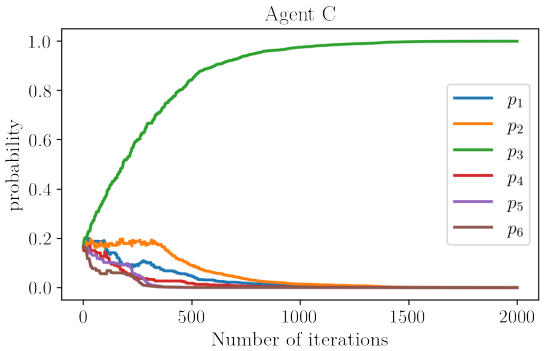
the other agent is sampled instead of chosen deterministically as discussed in Section 8.3. We note that our results from above is unchanged, the agents still converge to the social optimum – the algorithm is robust under sampling.



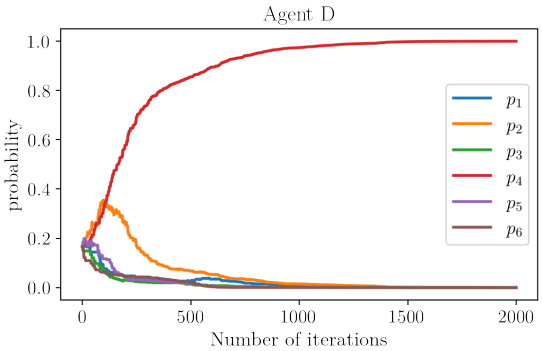
(a) Agent A's evolving probabilities.



(b) Agent B's evolving probabilities.

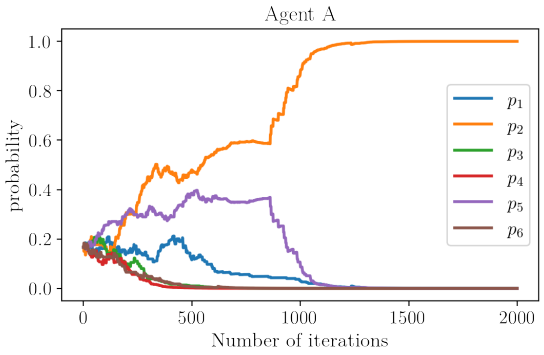


(c) Agent C's evolving probabilities.

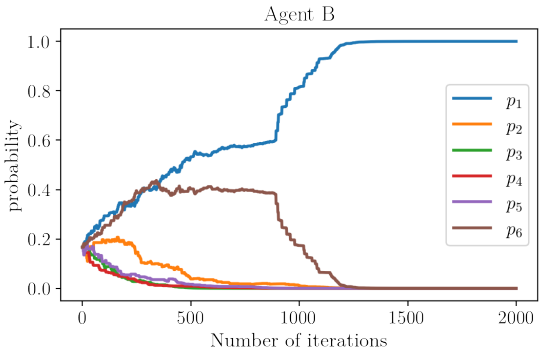


(d) Agent D's evolving probabilities.

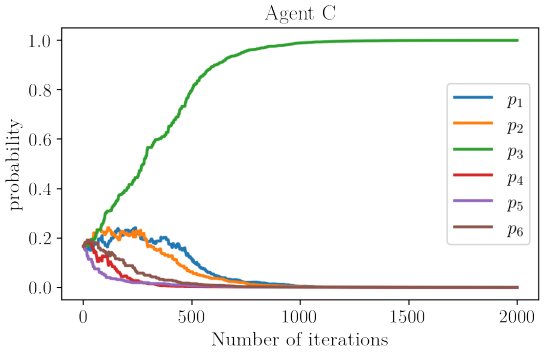
Figure I.6: Evolving probabilities for the multi-agent game by picking the other agent as the one who is most likely to choose the two actions.



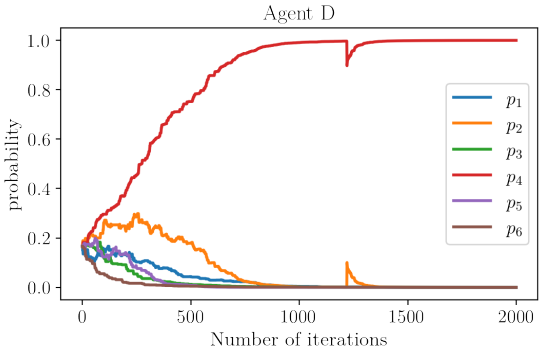
(a) Agent A's evolving probabilities.



(b) Agent B's evolving probabilities.



(c) Agent C's evolving probabilities.



(d) Agent D's evolving probabilities.

Figure I.7: Evolving probabilities for the multi-agent game when sampling for the other agent.

Appendix J

Further work on signatures and Neural CDEs

In this appendix, we give details of work inspired by rough path analysis and signatures. In Section J.1, we look at recent work done with co-authors to adapt previous neural CDEs, so that they are suitable for online prediction. Then in Section J.2, we look at applications in healthcare where signatures have been used as features in machine learning methods.

J.1 Online neural CDEs

Neural network structures based on differential equation concepts have been circulating in literature for the last few years, see for example [65], [39], and [56]. Within the machine learning community, Chen et al. [42] popularised these structures through the introduction of Neural ODEs with efficient gradient computation. Neural ODEs can be seen as a continuous time analogue to ResNets.

The limitations of using ODEs is that the dependence on the data is only through the initial condition. Neural CDEs [126] extend neural ODEs by allowing continuous dependence on the input data. However, in the approach for prediction tasks in [126], information on the whole timeseries was required at every timepoint, as the continuous control path was constructed with cubic spline interpolation (see Figure J.1).

Recent work with collaborators [125] extended the domain of application to online prediction tasks. In particular, as signatures are invariant to path reparameterisation, we can define a fictitious time (referred to as 'virtual time') across jumps when we receive new information, and therefore use a rectilinear interpolation approach at observation times (when new data arrive) to construct the control paths, see Figure J.1 (taken from [125]).

The illustration of this adaptation of neural CDEs to ensure any prediction is causal is given in Figure J.2. From the data, we use rectilinear interpolation (by adding a channel dimension for virtual time) to construct a continuous control path. We feed the control path into the neural CDE to learn a continuous hidden state, z, and at any time, a causal prediction can be made by applying a readout layer l on z.

J.2 Applications of signatures in data science for healthcare

As electronic patient health records become more widely adopted across the world, the amount of data available for research (under appropriate data access agreements) has also

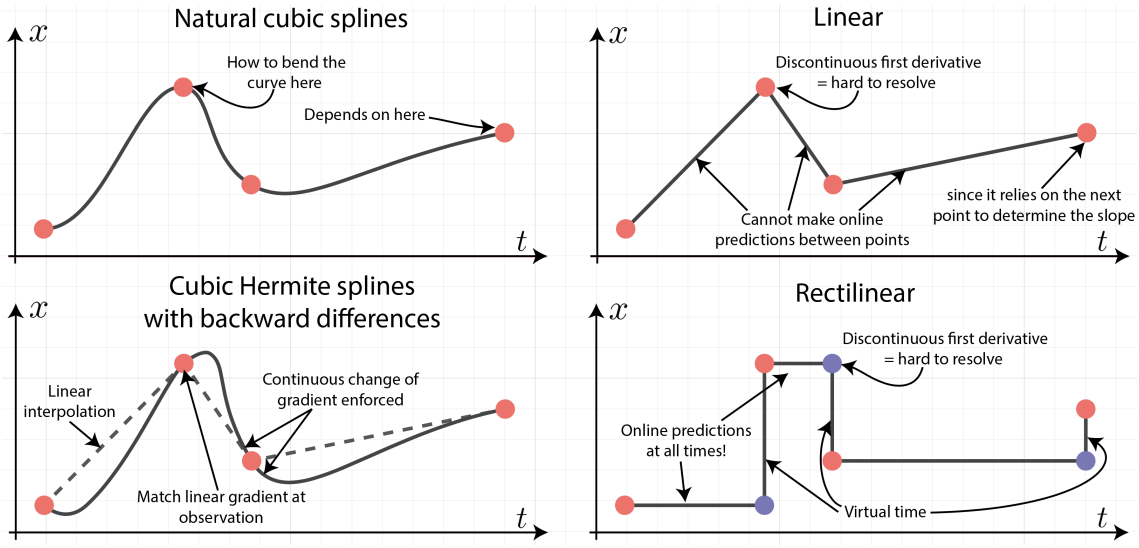


Figure J.1: Comparison of interpolation schemes for the construction of continuous control paths (taken from [125]).

become more abundant.

In medical healthcare data, it is important to consider the sequence and order of events (like the administration of specific drugs). The data may contain informed missingness, where the frequency/times of observation provide extra information (for example if the clinician is concerned about a patient, then the patient may be monitored more frequently).

Together with collaborators, I looked at two data science applications in healthcare. As part of these projects, we looked at individual data streams of patients, and whether we can effectively use path signatures as features to infer the medical conditions of patients. We outline the two applications below.

J.2.1 Early detection of sepsis

Sepsis is a major condition in healthcare. In 2017 alone, sepsis was estimated to affect nearly 50 million people worldwide, resulting in 11 million deaths [2]. Sepsis describes a very broad class of illnesses; like cancer, it encompasses varying symptoms across different organs. A common factor in sepsis is that major organs of the body shut down in order to preserve the most crucial functions. This can happen as a result of any infection.

A key research interest is to see if the onset of sepsis can be predicted accurately, and earlier. This formed the basis of the PhysioNet Computing in Cardiology Challenge 2019 [155], of which Morrill et al. [127] won with a LightGBM method utilising signature features.

In this work, we further investigate this early onset problem. We use the MIMIC-III database [95], which is a publicly available dataset gathered from the Beth Israel Deaconess hospital in Boston, containing de-identified patient-level data from over 40,000 patients that stayed in ICUs between 2001 and 2012.

Using path signatures in combination with LightGBM, CoxPHM, and LSTM, we obtain high performance models for the early prediction task. In addition, we highlight inconsistencies in the way that sepsis is identified from the data, and in fact this difference can be more pronounced than changing which model is used (LightGMB/CoxPHM/LSTM) [49].

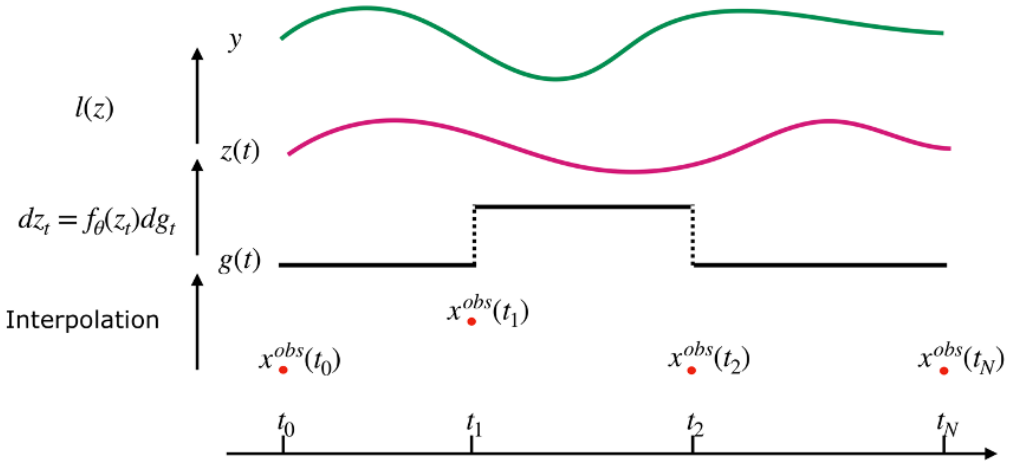


Figure J.2: Online neural CDE pipeline. From the observations, we use rectilinear interpolation to define a causal control path $g(t)$. This is used to control a latent variable $z(t)$ through the given CDE, where f is the neural network. Finally, a linear readout layer l maps z to the output variable y (which can be continuously defined at all times).

J.2.2 Early warning score differences in patients with and without covid

The aim of this project is to better understand the ways in which the presentation of deterioration may differ between those with and without COVID-19.

The data we use are private, pseudonymised data from two large UK hospital trusts, University Hospitals Birmingham NHS Foundation Trust (UHB) and University College London Hospitals (UCLH), which consist of over 100,000 patients.

The current national standard assessment for escalation of healthcare is NEWS2 [5]. This only looks at binned categories and map the incoming data into a very restricted set of discrete values.

We want to fully utilise the longitudinal data, and we extract signatures to use as features to predict deterioration. Since patient deterioration is not a well-defined concept, the target for classification is typically a proxy defined from a combination of ICU admission, mortality, and cardiac arrests. In our dataset, cardiac arrests cannot be inferred, and therefore we only use a combination of ICU admission and mortality. Given the specific interest in the prediction of deterioration in COVID-19 patients, we further designed a proxy for deterioration based on escalation in ventilation care.

By feeding the extracted signature features into LightGBM and CoxPHM models, we show in [51] that using a combination of signature features can outperform many popular early warning scores (including NEWS2). In addition, we show that even though COVID-19 is a new disease which was not used in the calibration of NEWS2, NEWS2 is able to better predict deterioration in these patients. This may be due to the fact that COVID-19 is more homogeneous as a condition, and the deterioration (for example fever, difficulty of breathing) are easy to identify from the data.