

ROBUST OPTIMAL CONTROL USING COMPUTATIONALLY
EFFICIENT DEEP REINFORCEMENT LEARNING
TECHNIQUES

by
Atish Dixit



Submitted for the degree of
Doctor of Philosophy

INSTITUTE OF GEOENERGY ENGINEERING
SCHOOL OF ENERGY, GEOSCIENCE, INFRASTRUCTURE AND SOCIETY
HERIOT-WATT UNIVERSITY

January 30, 2023

The copyright in this thesis is owned by the author. Any quotation from the report or use of any of the information contained in it must acknowledge this report as the source of the quotation or information.

Abstract

We investigate current challenges in the application of reinforcement learning (RL) to solve subsurface flow control problem which is the subject of intensive research in the field of reservoir management. In typical subsurface flow control problems, the system is partially observed because the data is often only available at well locations. Furthermore, the model parameters are highly uncertain as a result of the sparsity of available field data. As a result, we begin by presenting an RL framework to solve the stochastic optimal control for predefined model uncertainty and partially observable system. The numerical results are presented using two state-of-the-art model-free RL algorithms, proximal policy optimization (PPO) and advantage actor-critic (A2C), on two single-phase subsurface flow test cases representing two distinct flow scenarios. We identify that computational intractability is one of the major limitations for the proposed RL framework. This is because the model-free RL algorithms are by definition sample inefficient and require thousands if not millions of samples to learn optimal control policies. For subsurface control problems, this corresponds to performing a large number of simulations, which is computationally quite expensive. Our aim is to build a more generalized framework that can help alleviate this problem of computational complexity for the proposed RL framework. This is achieved by employing multiple levels of models. Here, the level refers to the accuracy or fidelity of the discretization of the domain grid of the underlying partial differential equations. We propose two distinct approaches that can be used in the most generalized manner. The first approach involves a more explicit modification of the proposed RL framework. In this approach, a multigrid framework is proposed that essentially takes advantage of the principles of sequential transfer learning. The second approach implicitly modifies the classical reinforcement learning framework itself to take advantage of information from lower-level models. This is achieved by modifying the classical framework of RL algorithms so that they use an approximate multilevel Monte Carlo estimates as opposed to Monte Carlo estimates of policy and/or value network objective functions.

Acknowledgements

I would like to thank my professors at the Center for Modeling and Simulation: Sukratu Barve, Mihir Arjunwadkar, Akanksha Kashikar and Valadi Jayaraman for the inspiring lectures during my masters studies. These lectures are the building blocks for my appreciation towards mathematics as the language of the universe.

I am deeply grateful to my supervisor Ahmed H. Elsheikh for granting me this wonderful opportunity to come and study at Heriot-Watt University, as well as for his support throughout this journey. I also thank my colleagues Shing Chan, Mark Ashworth and Amanzol Kubeyev for being good friends and their guidance through the PhD.

I would also like to acknowledge the Ali Danish Scholarship and the Engineering and Physical Sciences Research Council (EPSRC) for the financial support during my PhD work.

Last but not least, I would like to thank my friends and family for their relentless support at all stages of my life.

Research Thesis Submission

Please note this form should be bound into the submitted thesis.

| | | | |
|---|--|----------------|-----|
| Name: | Atish Narasinha Dixit | | |
| School: | School of Energy, Geoscience, Infrastructure and Society | | |
| Version: <i>(i.e. First, Resubmission, Final)</i> | Final | Degree Sought: | PhD |

Declaration

In accordance with the appropriate regulations I hereby submit my thesis and I declare that:

1. The thesis embodies the results of my own work and has been composed by myself
2. Where appropriate, I have made acknowledgement of the work of others
3. The thesis is the correct version for submission and is the same version as any electronic versions submitted*.
4. My thesis for the award referred to, deposited in the Heriot-Watt University Library, should be made available for loan or photocopying and be available via the Institutional Repository, subject to such conditions as the Librarian may require
5. I understand that as a student of the University I am required to abide by the Regulations of the University and to conform to its discipline.
6. I confirm that the thesis has been verified against plagiarism via an approved plagiarism detection application e.g. Turnitin.

ONLY for submissions including published works


Please note you are only required to complete the Inclusion of Published Works Form (page 2) if your thesis contains published works)

7. Where the thesis contains published outputs under Regulation 6 (9.1.2) or Regulation 43 (9) these are accompanied by a critical review which accurately describes my contribution to the research and, for multi-author outputs, a signed declaration indicating the contribution of each author (complete)
8. Inclusion of published outputs under Regulation 6 (9.1.2) or Regulation 43 (9) shall not constitute plagiarism.

* Please note that it is the responsibility of the candidate to ensure that the correct version of the thesis is submitted.

| | | | |
|-------------------------|---|-------|------------|
| Signature of Candidate: |  | Date: | 30/01/2023 |
|-------------------------|---|-------|------------|

Submission

| | |
|--|---|
| Submitted By <i>(name in capitals)</i> : | ATISH NARASINHA DIXIT |
| Signature of Individual Submitting: |  |
| Date Submitted: | 30/01/2023 |

For Completion in the Student Service Centre (SSC)


| | | | | | | |
|--|-----------|-----|----|----------|-----|----|
| Limited Access | Requested | Yes | No | Approved | Yes | No |
| <i>E-thesis Submitted (mandatory for final theses)</i> | | | | | | |
| Received in the SSC by <i>(name in capitals)</i> : | | | | Date: | | |

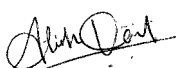
Inclusion of Published Works

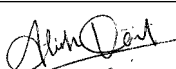
Please note you are only required to complete the Inclusion of Published Works Form if your thesis contains published works under Regulation 6 (9.1.2)

Declaration

This thesis contains one or more multi-author published works. In accordance with Regulation 6 (9.1.2) I hereby declare that the contributions of each author to these publications is as follows:

| | |
|------------------|---|
| Citation details | https://doi.org/10.1016/j.engappai.2022.105106 |
| Author 1 | Design theory and experiments for proposed framework and write the paper |
| Author 2 | Supervision for theory and design of experiments, guidance for writing the paper |
| Signature: |  |
| Date: | 17/10/2022 |

| | |
|------------------|---|
| Citation details | https://link.springer.com/article/10.1007/s11004-022-10033-x |
| Author 1 | Design theory and experiments for proposed framework and write the paper |
| Author 2 | Supervision for theory and design of experiments, guidance for writing the paper |
| Signature: |  |
| Date: | 17/10/2022 |

| | |
|------------------|---|
| Citation details | https://arxiv.org/abs/2210.08400 |
| Author 1 | Design theory and experiments for proposed framework and write the paper |
| Author 2 | Supervision for theory and design of experiments, guidance for writing the paper |
| Signature: |  |
| Date: | 17/10/2022 |

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Current approaches and their limitations | 2 |
| 1.2 | Why reinforcement learning? | 3 |
| 1.3 | Fundamentals of reinforcement learning | 3 |
| 1.4 | RL framework for optimal well control problem | 5 |
| 1.5 | Limitations of proposed RL framework | 6 |
| 1.5.1 | The explicit approach: adaptive multi-grid framework | 7 |
| 1.5.2 | The implicit approach: multi-level deep RL framework | 7 |
| 1.6 | Outline of the thesis | 8 |
| 1.6.1 | Outline of chapter 2 | 8 |
| 1.6.2 | Outline of chapter 3 | 9 |
| 1.6.3 | Outline of chapter 4 | 9 |
| 2 | Stochastic Optimal Well Control in Subsurface Reservoirs using Reinforcement Learning | 10 |
| 2.1 | Introduction | 11 |
| 2.2 | Methodology | 12 |
| 2.2.1 | Problem description for robust optimal well control | 12 |
| 2.2.2 | RL framework for robust optimal well control | 13 |
| 2.2.3 | RL algorithms | 17 |
| 2.2.3.1 | Advantage actor-critic algorithm | 17 |
| 2.2.3.2 | Proximal policy optimisation algorithm | 17 |
| 2.2.4 | Differential evolution algorithm | 19 |
| 2.2.5 | K-means clustering | 19 |
| 2.3 | Numerical experiments | 20 |
| 2.3.1 | model parameters | 20 |
| 2.3.2 | RL problem formulation: | 22 |
| 2.4 | Results and discussion | 23 |
| 2.5 | Conclusions | 31 |
| 3 | Robust Optimal Well Control using an Adaptive Multigrid Reinforcement Learning Framework | 33 |

| | | |
|----------|---|-----------|
| 3.1 | Introduction | 34 |
| 3.2 | Methodology | 35 |
| 3.2.1 | RL Framework | 37 |
| 3.2.2 | Learning Convergence Criteria | 38 |
| 3.2.3 | Adaptive Multigrid RL Framework | 40 |
| 3.3 | Case Studies | 43 |
| 3.3.1 | Uncertainty Distribution for Test Case 1 | 44 |
| 3.3.2 | Uncertainty Distribution for Test Case 2 | 45 |
| 3.3.3 | State, Action and Reward Formulation | 46 |
| 3.3.4 | Multigrid Framework Formulations | 47 |
| 3.4 | Results | 52 |
| 3.5 | Conclusion | 56 |
| 4 | A Multilevel Reinforcement Learning Framework for PDE based Control | 59 |
| 4.1 | Introduction | 60 |
| 4.2 | Background | 61 |
| 4.2.1 | Approximate Multilevel Monte Carlo estimation | 62 |
| 4.3 | Multilevel RL framework | 63 |
| 4.3.1 | Multilevel PPO algorithm | 64 |
| 4.3.2 | Multilevel PPO analysis methodology | 66 |
| 4.4 | Experiments | 69 |
| 4.4.1 | ResSim-v1 parameters | 69 |
| 4.4.2 | ResSim-v2 parameters | 70 |
| 4.4.3 | Reinforcement learning task | 71 |
| 4.4.4 | Multilevel framework formulation | 72 |
| 4.5 | Results | 74 |
| 4.5.1 | ResSim-v1 results | 75 |
| 4.5.2 | ResSim-v2 results | 77 |
| 4.5.3 | challenges and further research direction | 79 |
| 4.6 | Conclusions | 79 |
| 5 | Summarized Outlook of Proposed RL Frameworks for Subsurface Flow Control | 81 |
| 5.1 | Introduction | 82 |
| 5.1.1 | CLRM-based approach | 82 |
| 5.1.2 | RL-based approach | 82 |
| 5.1.3 | Advantages of RL-based approach | 83 |
| 5.2 | A summarized outlook of proposed RL frameworks | 84 |
| 5.3 | Guidelines for application of proposed framework | 86 |
| 5.4 | Future scope | 88 |
| 5.5 | Conclusion | 90 |

| | | |
|----------|---|------------|
| 6 | Conclusion and Future Work | 91 |
| 6.1 | Chapter 2: Stochastic optimal well control in subsurface reservoirs using reinforcement learning | 91 |
| 6.2 | Chapter 3: Robust optimal well control using an adaptive multigrid reinforcement learning framework | 92 |
| 6.3 | Chapter 4: Multilevel framework for deep reinforcement learning | 92 |
| 6.4 | Remarks and future directions | 92 |
| A | Appendices for Chapter 2 | 94 |
| A.1 | Definition of value and advantage functions | 94 |
| A.2 | Permeability uncertainty distribution parameters | 94 |
| A.3 | RL algorithm parameters | 95 |
| B | Appendices for Chapter 3 | 98 |
| B.1 | Cluster Analysis of Permeability Uncertainty Distribution | 98 |
| B.2 | Definitions of Value and Advantage Function | 99 |
| B.3 | Algorithm Parameters | 100 |
| C | Appendices for Chapter 4 | 103 |
| C.1 | Examples of objective functions for different deep RL algorithms | 103 |
| C.2 | Principle behind computational savings of MLMC estimator | 104 |
| C.3 | Implementation of multilevel PPO in stable baselines 3 | 105 |
| | C.3.1 Classical PPO implementation in stable baselines 3 | 105 |
| | C.3.2 Multilevel PPO implementation in stable baselines 3 | 107 |
| C.4 | Cluster analysis of permeability uncertainty distribution | 112 |
| C.5 | Algorithm Parameters | 113 |

List of Tables

| | | |
|-----|--|-----|
| 2.1 | Reservoir model parameters | 20 |
| 2.2 | number of simulation runs in each algorithm | 31 |
| 3.1 | Restriction operator function for simulation parameters | 40 |
| 3.2 | Reservoir model parameters | 44 |
| 3.3 | Grid fidelity factor and corresponding grid size | 47 |
| 3.4 | Multigrid framework experiments | 49 |
| 4.1 | grid size on each level | 72 |
| 4.2 | levels in each multilevel PPO experiment | 75 |
| 4.3 | parameters of multilevel PPO experiment for ResSim-v1 | 77 |
| 4.4 | parameters of multilevel PPO experiment for ResSim-v1 | 78 |
| A.1 | PPO algorithm parameters | 96 |
| A.2 | A2C algorithm parameters | 97 |
| A.3 | DE algorithm parameters | 97 |
| B.1 | PPO algorithm parameters | 100 |
| B.2 | DE algorithm parameters | 100 |
| C.1 | Objective function $\mathbb{E}_{s,a,r \sim p_\theta}[J(s, a, r; \theta, \Theta)]$ for different deep RL algorithms | 104 |
| C.2 | PPO algorithm parameters | 114 |
| C.3 | DE algorithm parameters | 114 |

List of Figures

| | | |
|------|---|----|
| 1.1 | A broad taxonomy of RL algorithms ¹ | 4 |
| 1.2 | The agent–environment interaction in RL framework | 6 |
| 2.1 | A typical agent-environment interaction in RL algorithms. state $s(x, t_m)$, action $a(x', t_m)$ and reward $r(t_m)$ are denoted with shorthand notations, s_m , a_m and r_m , respectively | 14 |
| 2.2 | optimal controls for complete control trajectory which refers to an episode in RL algorithms. state $s(x, t_m)$ and action $a(x', t_m)$ are denoted with shorthand notations, s_m and a_m , respectively | 14 |
| 2.3 | a wells spread choice of samples for some uncertainty distribution \mathcal{K} (depicted as one-dimensional for illustration purpose) to learn the robust optimal policy $\pi^*(a s; \mathbf{k})$. Members of \mathbf{k} are colored in white while the members of \mathbf{k}' are shown in grey. | 16 |
| 2.4 | the producers and injectors are highlighted with red and blue colors, respectively. parameters $(w, l_1$ and $l_2)$ for test case 1 log permeability are shown in fig (a), where high permeability channel is colored in grey. | 21 |
| 2.5 | clustering of log-permeability fields (unit: mD) for test cases 1 and 2 | 23 |
| 2.6 | log-permeability plots (unit: mD) for evaluation data for test cases 1 and 2 | 24 |
| 2.7 | RL algorithm agent-environment interaction schematics to learn robust optimal well control policy | 24 |
| 2.8 | Test case 1: monitoring plots for average training return $\mathbf{R}^{\pi(a s; \mathbf{k} \Rightarrow \mathbf{k})}$ (on left) and evaluation return $\mathbf{R}^{\pi(a s; \mathbf{k} \Rightarrow \mathbf{k}')$ (on right) for learning process in PPO, A2C and frozen PPO. The evaluation return value is compared with the optimisation results obtained using DE | 26 |
| 2.9 | Test case 1: comparison of optimum recovery factor (in %) for each permeability value from evaluation vector \mathbf{k}' . Results of of PPO, A2C and DE are compares with base control actions (all control valves equally open) results | 26 |
| 2.10 | Test case 1: optimal well controls for permeability values \mathbf{k}'_3 and \mathbf{k}'_5 (illustrated with saturation contour plots). Producer and injector flow controls are indicated with red and blue circles, respectively. Values of flow controls are proportional to the circle radius. | 27 |

| | | |
|------|--|----|
| 2.11 | Test case 2: monitoring plots for average training return $\mathbf{R}^{\pi(a s;\mathbf{k}\Rightarrow\mathbf{k})}$ (on left) and evaluation return $\mathbf{R}^{\pi(a s;\mathbf{k}\Rightarrow\mathbf{k}')}$ (on right) for learning process in PPO and A2C. The evaluation return value is compared with the optimisation results obtained using DE | 29 |
| 2.12 | Test case 2: comparison of optimum recovery factor (in %) for each permeability value from evaluation vector \mathbf{k}' . Results of of PPO, A2C and DE are compared with base control actions (all control valves equally open) results | 29 |
| 2.13 | Test case 2: optimal well controls for permeability values \mathbf{k}'_9 and \mathbf{k}'_{12} (illustrated with saturation contour plots). Producer and injector flow controls are indicated with red and blue circles, respectively. Values of flow controls are proportional to the circle radius. | 30 |
| 3.1 | Plot of policy returns versus number of training episodes: a illustrates effect of δ on convergence criteria and b illustrates effect of n on convergence criteria | 39 |
| 3.2 | Illustration for the restriction operator Φ_β (a) and prolongation operator Φ_β^{-1} (b) for a parameter x | 41 |
| 3.3 | A typical agent-environment interaction in the proposed multigrid RL framework | 42 |
| 3.4 | Schematic of the spatial domain for test case 1 (a) and 2 (b) | 44 |
| 3.5 | Comparison of recovery factor estimates with $\beta = 1$, $\beta = 0.5$ and $\beta = 0.25$ for test case 1 (a) and test case 2 (b) | 48 |
| 3.6 | Effect of grid fidelity factor β on the environment for test case 1: a on a sample of log-permeability (unit: mD), b on corresponding saturation and c on simulation run time | 50 |
| 3.7 | Effect of grid fidelity factor β on the environment for test case 2: a on a sample of log-permeability (unit: mD), b on corresponding saturation and c on simulation run time | 51 |
| 3.8 | Plots of policy return versus number of episodes for test case 1 | 53 |
| 3.9 | Evaluation of learned policies for test case 1: a evaluation samples of log-permeability distribution \mathcal{G}_1 shown with contour plots (unit: mD), b recovery factor (in % format) versus evaluation sample index (from a) plot | 54 |
| 3.10 | Illustration of learned optimal control policies for test case 1 using saturation contour plots | 55 |
| 3.11 | Plots of policy return versus number of episodes for test case 2 | 55 |
| 3.12 | Evaluation of learned policies for test case 2: a evaluation samples of log-permeability distribution \mathcal{G}_2 shown with contour plots (unit: mD), b recovery factor (in % format) versus evaluation sample index (from a) plot | 57 |
| 3.13 | Illustration of learned optimal control policies for test case 2 using saturation contour plots | 58 |
| 4.1 | schematics of rollouts for a policy iteration | 65 |
| 4.2 | schematic of the spatial domain Ω | 70 |

| | | |
|------|---|-----|
| 4.3 | example policy visualization for ResSim-v1 | 72 |
| 4.4 | example of policy visualization for ResSim-v2 | 73 |
| 4.5 | environment levels for ResSim-v1 | 74 |
| 4.6 | environment levels for ResSim-v2 | 75 |
| 4.7 | comparison of Monte Carlo and multilevel Monte Carlo estimate of PPO objective function for ResSim-v1 | 75 |
| 4.8 | MLMC analysis results for ResSim-v1 | 76 |
| 4.9 | multilevel PPO results for ResSim-v1 | 77 |
| 4.10 | comparison of Monte Carlo and multilevel Monte Carlo estimate of PPO objective function for ResSim-v2 | 78 |
| 4.11 | MLMC analysis results for ResSim-v2 | 78 |
| 4.12 | multilevel PPO results for ResSim-v2 | 79 |
| 5.1 | Approaches to solving optimal well control problems | 83 |
| 5.2 | Evolution of proposed RL frameworks | 85 |
| A.1 | Value for low (-2) and high (5.5) log permeability in test case 1 was chosen from the SPE-10 model 2 Upperness permeability distribution peaks | 95 |
| A.2 | Mean of (2.4) log permeability in test case 2 was chosen from the SPE-10 model 2 Tarbert case data. Log permeability distribution chosen in test case 2 is superpositioned with Tarbert permeability distribution (shown with thick black line) for comparison | 96 |
| A.3 | learning plot range over three distinct seed values for test case 1 and 2 | 97 |
| B.1 | Log-permeability plots for training data of test case 1 and 2: a and b illustrate clustering for \mathcal{G}_1 and \mathcal{G}_2 distribution samples | 99 |
| B.2 | Learning plots with seed 1 (a), 2 (b) and 3 (c) for test case 1 | 101 |
| B.3 | Learning plots with seed 1 (a), 2 (b) and 3 (c) for test case 2 | 102 |
| C.1 | A typical agent-environment interaction for classical framework | 105 |
| C.2 | Object-oriented design for the stable baselines implementation of PPO algorithm | 108 |
| C.3 | A typical agent-environment interaction for an environment on level l synchronized with environment on level $l - 1$ | 109 |
| C.4 | RolloutBufferArray (on left) and SyncRolloutBufferArray (on right). SyncRolloutBuffer ^{l} consists of synchronized data of RolloutBuffer ^{l} with level l to a level $l - 1$. Each buffer with level l consists of $N \times T_l$ rows of data in $[s, a, r, d, V, L_{\text{old}}, R, A]$ format. | 109 |
| C.5 | batch_array which is achieved from GetBatches function. It consists of in total NT_l/M_l batches as shown with the columns of the array. Each such batch consists of L batches from RolloutBuffers (denoted by batch ^{l}) and SyncRolloutBuffers (denoted by syncBatch ^{$l-1$}). batch ^{l} and SyncBatch ^{$l-1$} consists of M^l rows of data in the format, $[o, a, V, L_{\text{old}}, R, A]$. | 110 |

C.6 Object-oriented design for the stable baselines implementation of multilevel PPO algorithm. The updated (from classical PPO implementation) definitions of functions and classes are highlighted in red colour. 112

C.7 clustering visualization for permeability samples 113

Chapter 1

Introduction

Fluid flow control in subsurface reservoirs has many engineering applications ranging from the financial aspects of efficient hydrocarbon production to the environmental problems of contaminated removal from polluted aquifers. Mathematically, it is formulated as an optimal control problem that deals with finding a *control* for a *dynamical system* over a period of time so that an *objective function* is optimized. For subsurface flow control problems, *control* refers to the flow through injector and/or producer wells, *dynamical system* refers to a set of partial differential equations (PDE) that describe the flow through the porous medium of the reservoir, and *objective function* refers to the sweep efficiency of the injected fluid. For efficient hydrocarbon production, the objective function is often extended to formulate the net present value. In solving the above-mentioned subsurface flow control problems, reservoir engineers often need to account for an incomplete description of subsurface properties. This leads to two main challenges. First, the parameters of the said PDEs need to be modeled with uncertainty to reflect this incompleteness in the available reservoir data. Furthermore, throughout the reservoir life cycle, the data are only partially observable; often only at the well locations.

In recent years, many advances have been made by major research groups like Google DeepMind and OpenAI which produced astonishing results using reinforcement learning (RL). The successful results of RL algorithms such as AlphaZero [1] and AlphaGo [2] in strategy board games showed that RL algorithms are the best candidates to solve problems in highly dynamical systems. So far, research on the application of RL in optimal control problems is advancing rapidly in fields such as manufacturing [3], energy [4] and fluid dynamics [5]. However, research focused on developing RL strategy for large-scale simulations based on assessment of its robustness against the uncertainties is still an open area, especially for cases where model uncertainties has a substantial effect on the optimal control. In this thesis, we study the application of reinforcement learning to solve subsurface flow control problems. In particular, we begin by introducing an RL framework to solve the stochastic optimal well control problem for partially observable simulation data (Chapter 2). We acknowledge the computational cost used by simulation runs as the major limitation of the proposed RL algorithm. As a result, we propose an explicit RL approach (Chapter 3) and an implicit RL approach (Chapter 4) to

alleviate the overall computational cost of the proposed RL framework. A final summary of the thesis is presented in Chapter 5.

1.1 Current approaches and their limitations

In petroleum engineering, research studies for solving real-time subsurface flow control problems fall under the category of closed-loop reservoir management. In a nutshell, the closed-loop reservoir management (CLRM) technique enables a dynamic and real-time optimal production schedule under existing reservoir conditions to be achieved by adjusting the injection and production strategies. CLRM can be further divided into two main research subcategories: automatic history matching and reservoir control optimization.

Automatic history matching is the sequential modeling update method in which estimates of uncertain reservoir properties are constantly updated according to available production measures at the time. It has been studied since the 1960s [6–8], but it remains a very relevant and challenging research topic. Existing history matching methods can be broken down into two categories. The first category is gradient-based, including the finite-difference derivative approximation, adjoining gradient-based methods, and gradient simulation methods. The other category is based on gradient-free optimization, such as simultaneous perturbation stochastic approximation, genetic algorithm, particle swarm optimization, and pattern search methods. [9] provide a thorough summary on recent progress in automatic history matching.

Reservoir control optimization is a full or partial automation process that aims to optimize operating parameters to maximize total production throughout the life cycle of a reservoir by optimizing operating parameters. The origin of solving reservoir control problems using optimization theories can be traced back to [10], where they used linear programming to maximize the net present value of production for the homogeneous reservoir. However, most papers published before the 1980s did not pay enough attention to optimization algorithms, and successful applications were very rare [11–13]. With advances in optimization algorithms and computing power, research has grown considerably since the 1980s [14–17]. Most optimization methods solve optimal control problems by finding optimal control values using a parameter search approach, either gradient-based [18–21] or gradient-free [22, 23]. The main limitations of such direct optimization methods is that they are inherently unsuitable for stochastic problems. Studies conducted by [24] or [25] are examples of ad hoc modifications of these optimization methods to account for model uncertainties. As a result, we ask whether we can employ a research methodology that sidesteps history matching and, at the same time, provides a robust optimal solution to subsurface flow control problems against the model uncertainties.

1.2 Why reinforcement learning?

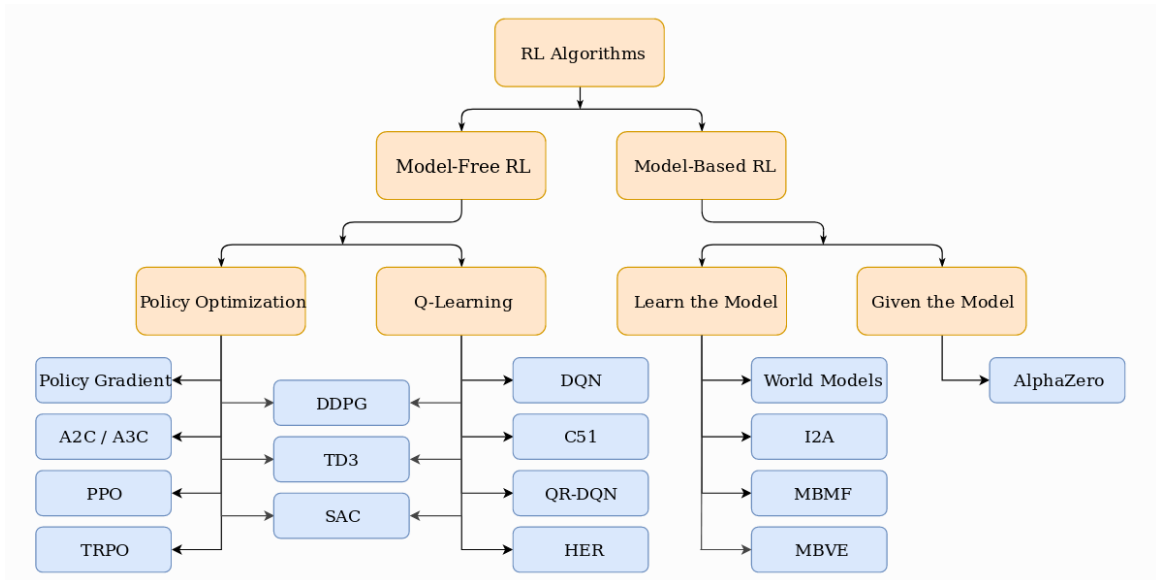
Reinforcement learning provides a framework for solving optimal well control problems that bypass the conventional two-stage approach in CLRM. Since the optimal control policy learned to use this approach is modeled as a probability distribution, the uncertainty in the model parameters is thoroughly considered when learning the policy. As a result, the model calibration step is redundant in the optimization process. However, before we dive into reinforcement learning methods for the problem at hand, let us also discuss another traditional approach called optimal control theory.

Most optimal control theory research for solving optimal control problems involves, in principle, finding an optimal control by deriving Pontryagin’s maximum principle or by solving the Hamilton–Jacobi–Bellman equation. These classical strategies are offline and require complete knowledge of system dynamics, which makes them unsuitable for dynamical systems with uncertainties [26]. Recently, model predictive control (MPC), a feedback control based on real-time optimization, has attracted increasing attention in the research of stochastic optimal control [27–29]. Alternatively, optimal control problems can also be solved using reinforcement learning (RL) approaches. Koryakovskiy et al. [30] provide a benchmark study for the comparison between model predictive control and model-free reinforcement learning approaches, where the authors show that RL results are comparable to MPC. Furthermore, after a certain break-even point, model-free reinforcement learning is shown to perform better than non-linear model predictive control with an inaccurate model. Furthermore, unlike MPC, RL provides an additional advantage of generality, since it does not require complete knowledge of the model dynamics. So far, research on the application of RL in optimal control problems is advancing rapidly in fields such as manufacturing [3], energy [4] and fluid dynamics [5].

1.3 Fundamentals of reinforcement learning

The main characters of RL are agents and the environment. The environment is the world where an agent lives and interacts. At each interaction step, the agent sees (arguably partially) an observation of the world’s state and decides what action to take. The environment changes as the agent acts on it, but this environment can also change itself. Agents also receive reward signals from their environment that tell them exactly how good or bad the state of the world is. The goal of the agent is to maximize its cumulative reward, called the return. Reinforcement learning algorithms basically describe the ways in which the agent can learn behaviors to achieve its goal.

Reinforcement learning differs from other machine learning methods in several ways. The data used to train the agent are collected through interactions with the environment by the agent itself (compared to supervised learning where you have, for instance, a fixed dataset). If an agent collects poor quality data (e.g., trajectories without rewards), then it will not improve and will continue to accumulate bad trajectories. For this reason, among others, RL results

Figure 1.1: A broad taxonomy of RL algorithms ¹

often vary from one run to another (i.e., when only the seed of the pseudo-random generator changes). As a result, it is common practice to perform several runs of RL algorithms to obtain quantitative results. Good results in RL are generally dependent on finding appropriate hyperparameters that correspond to most stable results.

A review of various deep RL algorithms is illustrated in Figure 1.1 as a taxonomy. Depending on whether the agent has access to (or is learning) an environment model, the algorithms are divided into two parts: model-free and model-based algorithms. The model of the environment refers to a function that predicts state transitions and rewards. Model-free algorithms can be divided into two main strategies: policy optimization and Q-learning. A2C/A3C [31] and PPO [32] are examples of such policy optimization algorithms where the policy parameters are optimized directly or indirectly. On the other hand, Q-learning methods learn an approximator for the action value function, for example, the DQN algorithm [33]. Alternatively, model-based RL have access to an approximate model for the environment and that allows the agent to plan ahead, which could be a main advantage in some cases. A particular example of this approach is AlphaZero [1]. Model-based algorithms can be broadly classified into three categories. First, pure planning algorithms that use techniques such as model predictive control, e.g., [34]. AlphaZero [1] and ExIt [35] algorithms are examples of the second category, expert iteration, where actions are sampled from its current policy. Finally, the data augmentation method uses a model-free RL algorithm that augments real experiences with fictitious ones, for example, world models [36].

The optimality of the model-based algorithms' policies is highly dependent on the accuracy of model learning process. As a result, any error in model learning is subject to a compounding increase in error in the policy learning process. Lambert et al. [37] present a study that highlights a fundamental issue of objective mismatch in model-based algorithms. In this study,

¹Image source for Figure 1.1: https://spinningup.openai.com/en/latest/spinningup/rl_intro2.html

the objective mismatch corresponds to the mismatch between local (i.e., one-step ahead dynamics) accuracy of the learned model dynamics and the overall improvement in performance on a downstream control task. Furthermore, the model-based algorithms are computationally more demanding, since they also involve collecting the data from environment transitions and training a new model to learn the transition function of the environment. However, there is a trade-off in sample efficiency once the model is learned. As a result, despite its sample inefficient nature, most state-of-the-art results in RL applications are presented using model-free algorithms. For these reasons, we decide to choose the model-free algorithm as a suitable candidate to solve the subsurface flow control problem. In other words, the model dynamics is performed directly by solving simulations of PDEs for the subsurface flow problem. While modeling the subsurface control problems using RL algorithms, we also have to consider the type of control variables in this problem. In these problems, the well controls should be treated as continuous variables in order to accurately reflect the on-field reservoir scenario. For this reason, the choice of RL algorithms must also provide provision to model the optimal policy in terms of continuous controls. In model-free reinforcement learning algorithms, actor-critic algorithms are best suited for such cases. As a result, we demonstrate our first RL framework (presented in Chapter 2) using two actor-critic algorithms called as A2C and PPO. Note that A2C and PPO algorithms come from the family of REINFORCE algorithms where PPO is the most state-of-the-art algorithm which improves the stability of the policy learning process in algorithms like A2C. As a result, we continue our demonstrations in Chapter 3 and 4 using PPO algorithm.

1.4 RL framework for optimal well control problem

Reinforcement learning problems involve learning what to do, how to map situations to actions, and maximize a numerical reward signal. Learning is achieved by using episodic simulations of agent-environment interactions. In the context of optimal well-control problems, the agent refers to a controller which follows and eventually learns the optimal policy (mapping from state to action). The environment consists of model dynamics (which are modeled as reservoir simulation) and a reward system, which computes sweep efficiency at a certain time of the episode (a.k.a., reward). The episode refers to reservoir simulation for the whole reservoir life cycle which is divided into a certain number of control steps. Figure 1.2 illustrates a closed-loop schematic of the agent-environment interaction at a certain control step t . Here, state refers to certain variables (e.g. saturation, pressure, etc.) of the model, while action refers to flow control value at the injector and producer well location. RL algorithms often involve a large number of such agent-environment interactions, which are eventually used to learn and optimize the controller policy. The stochasticity of the model dynamics is represented with the uncertainty of the model parameters. This is done by representing this parameter with a predefined probability distribution. In practice, such a distribution can be modeled as a posterior distribution with prior observations from on-field data. The policy is learned using

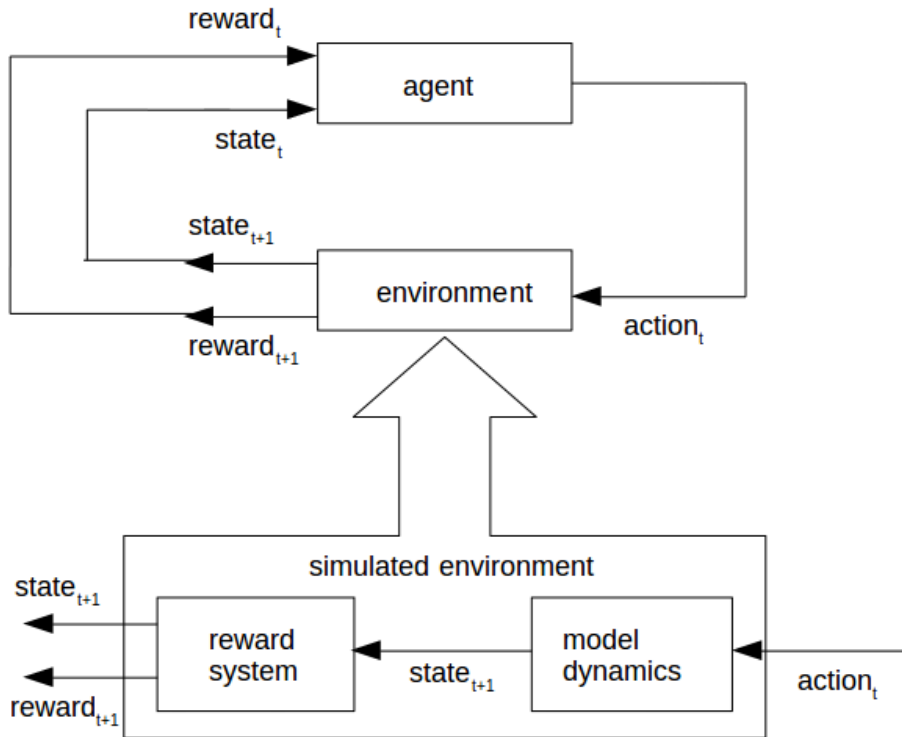


Figure 1.2: The agent–environment interaction in RL framework

certain distribution samples that are selected using a domain randomization technique, which involves clustering of the distribution domains. Additionally, the robustness of the policy is evaluated using new distribution samples that were not seen at the time of the policy learning process. We present this framework using two state-of-the-art model-free RL algorithms called the advantage actor-critic algorithm [38] and the proximal policy optimization algorithm [39].

1.5 Limitations of proposed RL framework

For demonstrated case studies with single-phase flow simulations, we found that it took around 60 to 200 thousand simulation runs to learn the optimal policy. This took from two to ten hours of wall clock time to complete the reinforcement learning process using our available computational resources (a desktop with 64 CPUs and two GPUs). Although this does not seem extremely time-consuming, note that this time is for simplified two-dimensional cases. If we extend these case studies to large-scale simulations (for instance, two- or three-phase flow simulations in the 3D domain), it would be quite intractable to use the proposed RL framework. The computational cost per se has always been a common challenge in reinforcement learning (RL) methods. For instance, the RL-based chess engine AlphaGo Zero, by Google cost worth \$350 millions of computational resources for its training [40].

The easiest way to deal with this limitation is to create a data-based surrogate model that mimics the physics of the underlying PDEs (as demonstrated by [41]). However, this comes with

an extra cost for the process of generating data and fitting a data-based model. Furthermore, much like model-based reinforcement learning methods, the optimal learned policy is also highly dependent on the accuracy of the learned model. Not to mention that such a method requires building a data-based model for every new problem, making it a very ad hoc approach.

In this thesis, our aim is to build a more generalized framework that can help alleviate the computational complexity of the proposed RL framework. Essentially, our aim is to use multiple levels of models that describe the said PDEs. Here, the level refers to the accuracy or fidelity of the discretization of the domain grid. We propose two such distinct approaches that can be used in the most generalized manner. The first approach involves a more explicit modification of the proposed RL framework. We propose a multigrid framework that takes advantage of the principles of sequential transfer learning. The second approach implicitly modifies the reinforcement learning framework to take advantage of the information from the lower-level models. This is done by modifying the RL algorithms so that they use multilevel Monte Carlo estimates as opposed to Monte Carlo estimates in their classical form. We delineate these approaches in the following subsections.

1.5.1 The explicit approach: adaptive multi-grid framework

To address the computational complexity bottleneck of the proposed RL framework, we introduce an adaptive multigrid RL framework that is inspired by the principles of multigrid geometric methods used in iterative numerical algorithms. In this framework, RL control policies are initially learned using computationally efficient low-fidelity simulations using coarse grid discretization of the underlying PDEs. Subsequently, the simulation fidelity is increased in an adaptive manner towards the highest fidelity simulation that corresponds to the finest discretization of the model domain. The proposed framework is demonstrated using a state-of-the-art, model-free policy-based RL algorithm, namely the Proximal Policy Optimization (PPO) algorithm.

1.5.2 The implicit approach: multi-level deep RL framework

We identify the main source of computational cost in reinforcement learning, which corresponds to the transition function of the model dynamics. This is especially problematic where problems such as optimal well control are represented with PDEs. This is because in such cases the transition function often involves solving a large-scale discretization of the said PDEs. We propose a multi-level RL framework in order to ease this cost by exploiting some sublevel models that correspond to coarser scale discretization (i.e. multi-level models). This is done by formulating multilevel Monte Carlo estimates (adopted from [42]) of the policy and/or value network objective function instead of Monte Carlo estimates, as done in the classical framework. As a demonstration of this framework, we present a multilevel version of the proximal policy optimization (PPO) algorithm.

1.6 Outline of the thesis

Reinforcement learning is a promising tool to solve the problem of subsurface flow control. Especially because of its robustness against uncertainties and partial observability of the model, which are common challenges in the modeling of reservoir simulations. Naturally, we began our research by creating a robust RL framework to solve optimal well control problems. We present this framework in the second chapter of the thesis. As is usual in many simulation-based RL methods, we notice the overbearing problem with the computational cost of performing model dynamics in RL process. Because of such a generality of the problem at hand, we emphasized creating a more general framework, which not only solves the issue of optimal well control problem but can also be applied to other simulation-based optimal control problems. In the third chapter of the thesis, we propose a generalized adaptive transfer learning framework to alleviate the overall computational cost of the proposed RL framework. Albeit the attempt towards attaining the generality in this framework, we note that the framework is designed with problem-specific hyperparameters, which are used to test the convergence criteria of transfer learning. As we move further in our research endeavors, we investigate an even more generalized solution in the fourth chapter. We propose a fundamental change in the way data generation takes place in the classical RL framework. This attempt leads us to develop a novel synchronized multilevel data generation method which is used in the policy learning process. In contrast to the framework presented in the third chapter, no additional hyperparameters are used in this proposed multilevel RL framework. The only change in the input parameters refers to a multilevel representation of the parameters of the classical algorithms. In the presented case studies, we show only a multilevel version of the PPO algorithm. However, because of the foundational generality in this approach, one can also produce multilevel versions of other RL algorithms, which can be effectively used in simulation-based optimal control problems. As a result, we can observe that although we start the thesis specifically by keeping subsurface flow control problem in mind, it eventually evolves in a more generalized approach towards tackling the challenge of computational complexity for general simulation-based RL problems.

The following is a summary of each of the remaining chapters. Note that each chapter intends to be self-contained, as they were initially written as separate papers.

1.6.1 Outline of chapter 2

In this chapter, we utilize a model-free RL algorithm to solve a simulation-based robust optimal control problem where the model information is assumed to be uncertain and partially available. We begin with the definition of the problem, where the control variables of the well are optimized to maximize the sweep efficiency of the injection fluid throughout the reservoir life cycle. The assumption of partially available model information is based on the fact that reservoir field data are generally only available at well locations. We consider the reservoir permeability field as an uncertain model parameter. We present two test cases that underscore the effect of parameter uncertainty on optimal controls. For ease of execution and demonstration, we

represent the optimal well control problem with an advection equation for tracer flow through porous media, in which an uncertain parameter, permeability, is treated as a random variable. In the first test case, we used a permeability field distribution where the location and orientation of a linear high-permeability channel were uncertain. The second test case uses a spatially correlated permeability field to represent a smoother permeability field, where the logarithmic permeability field is modeled with a conditional Gaussian distribution with an exponential kernel. We employ proximal policy optimization (PPO) and advantage actor-critic (A2C) algorithms to learn the optimal well control policy in order to maximize the sweep efficiency in the domain. RL policies are learned using a selected number of realizations of the uncertainty distribution, and its robustness is evaluated by applying the learned policy on a set of unseen realizations during training, drawn from the same distribution.

1.6.2 Outline of chapter 3

In this chapter, we propose an adaptive multigrid framework which uses the principles of sequential transfer learning from models with low to high-fidelity grid representations of the environment. Knowledge transfer is done in the form of a policy for a model-free, on-policy algorithm called proximal policy optimization (PPO). This is an extension of the RL framework presented in Chapter 2, where we propose a framework that could significantly reduce the overall computational cost of the reinforcement learning process. We consider the same problem which consists of optimizing the control variables such as valve openings of wells in order to maximize sweep efficiency of injector fluid throughout the reservoir life, and the reservoir permeability field is parameterized with a predefined uncertainty distribution. Two test cases, both representing distinct model parameter uncertainty and control dynamics, are used to demonstrate the computational gains of using the proposed multigrid framework.

1.6.3 Outline of chapter 4

We use a similar set of problems and case studies as used in previous chapters to demonstrate the results of the proposed multilevel RL framework in Chapter 4. We begin by explaining the background and general anatomy of the classical RL framework and deep RL algorithms. In addition, the multilevel version of this classical framework is presented along with the abstract outline of the proposed multilevel RL algorithms. Furthermore, we present a multilevel representation of the classical PPO algorithm. The effectiveness of multilevel PPO is presented by comparing the computational complexity of classical and multilevel PPO algorithms. We expand on the results of the multilevel PPO algorithm by performing a thorough analysis of Monte Carlo and the multilevel Monte Carlo estimation process.

Finally, we provide a summarized outlook on the presented frameworks in Chapter 5 and conclude our thesis in Chapter 6.

Chapter 2

Stochastic Optimal Well Control in Subsurface Reservoirs using Reinforcement Learning

We present a case study of model-free reinforcement learning (RL) framework to solve stochastic optimal control for a predefined parameter uncertainty distribution and partially observable system. We focus on robust optimal well control problem which is a subject of intensive research activities in the field of subsurface reservoir management. For this problem, the system is partially observed since the data is only available at well locations. Furthermore, the model parameters are highly uncertain due to sparsity of available field data. In principle, RL algorithms are capable of learning optimal action policies — a map from states to actions — to maximize a numerical reward signal. In deep RL, this mapping from state to action is parameterized using a deep neural network. In the RL formulation of the robust optimal well control problem, the states are represented by saturation and pressure values at well locations while the actions represent the valve openings controlling the flow through wells. The numerical reward refers to the total sweep efficiency and the uncertain model parameter is the subsurface permeability field. The model parameter uncertainties are handled by introducing a domain randomisation scheme that exploits cluster analysis on its uncertainty distribution. We present numerical results using two state-of-the-art RL algorithms, proximal policy optimization (PPO) and advantage actor-critic (A2C), on two subsurface flow test cases representing two distinct uncertainty distributions of permeability field. The results were benchmarked against optimisation results obtained using differential evolution algorithm. Furthermore, we demonstrate the robustness of the proposed use of RL by evaluating the learned control policy on unseen samples drawn from the parameter uncertainty distribution that were not used during the training process.

2.1 Introduction

Optimal control problem involves finding controls for a dynamical system such that a certain objective function is optimized. Traditionally, most research for solving the optimal control problems for non-linear dynamical systems uses optimal control theory which, in principle, finds optimal control by deriving Pontryagin’s maximum principle or by solving the Hamilton–Jacobi–Bellman equation. These classical strategies are offline and require a complete knowledge of system dynamics making them unsuitable for dynamical systems with uncertainties [26]. Recently, model predictive control (MPC) – a feedback control based on real-time optimisation – has attracted increasing attention in stochastic optimal control research [27–29]. Alternatively, optimal control problems could also be solved using reinforcement learning (RL) approaches. Koryakovskiy et al. [30] provide a benchmark study for comparison between model predictive control and model-free reinforcement learning approaches where the authors show that RL results are comparable to MPC. Further, after a certain break-even point in model uncertainties, model-free reinforcement learning is shown to perform better than nonlinear model predictive control with an inaccurate model. In other words, model-free reinforcement learning is found to be more advantageous compared to MPC when more uncertainties were introduced in the model. Furthermore, as opposed to MPC, RL provides an extra advantage of generality since it doesn’t need complete knowledge of model dynamics. So far, research on application of RL in optimal control problems is advancing rapidly in fields like manufacturing [3], energy [4] and fluid dynamics [5]. However, research focused on developing RL strategy based on assessment of its robustness against the uncertainties is still an open area, especially for cases where model uncertainties has a substantial effect on the optimal control.

In this study, we utilize a model-free RL algorithm to solve simulation-based robust optimal control problem where the model information is assumed to be partially available. *Robust optimal well control* in petroleum reservoir management [24, 43, 44] forms a perfect candidate for such problem. In this problem, the reservoir well control variables are optimized in order to maximize the sweep efficiency of injection fluid throughout the reservoir life cycle. The assumption of partially available model information is based on the fact that the reservoir field data is generally only available at well locations. We consider reservoir permeability field as an uncertain model parameter. Results are demonstrated for two state of the art model-free RL algorithms: proximal policy optimisation [39] and advantage actor-critic [38]. Although we utilize robust well control in the current study, the presented techniques are general and is applicable to a wide range of simulation-based nonlinear optimal control problems.

We designed two test cases that underscore effect of parameter uncertainty on optimal controls. For ease of execution and demonstration, we represent the optimal well control problem with an advection equation for tracer flow through porous media in which uncertain parameter, permeability, is treated as a random variable. In the first test case, we use a permeability field distribution where the location and orientation of a linear high permeability channel is uncertain. The second test case uses a spatially correlated permeability field to represent

smoother permeability field where the log-permeability field is modeled with a conditional Gaussian distribution with an exponential kernel. We employed proximal policy optimisation (PPO) and advantage actor-critic (A2C) algorithms to learn the optimal well control policy in order to maximize the sweep efficiency in the domain. The RL policies are learned using a selected number of realizations of uncertainty distribution and its robustness is evaluated by applying the learned policy on a set of unseen realizations during training, drawn from the same distribution.

The outline of the rest of this paper is as following: Section 2.2 provides the problem description and how RL algorithms are utilized to solve robust optimal well control. Information such as algorithms and methodologies used in this paper are also briefed in this section. Section 2.3 details the model parameters for test cases chosen for this study. Further, the approach used for problem formulation for RL algorithms is explained. Results for the given test cases are presented in section 2.4. Finally, section 2.5 concludes with the research study summary and few future research directions.

2.2 Methodology

The process of single-phase flow in porous media is of importance to a variety of engineers and scientists who are concerned with problems ranging from the financial aspects of oil movement in petroleum reservoirs to the social problems of groundwater flows in polluted aquifers [45]. We demonstrate the proposed techniques in the context of subsurface flow in subsurface reservoirs using an advection equation for tracer flow through porous media. In this dynamical system, the *tracer* enters the domain and pushes the *non-traced fluid* out of the domain. Flow in and out of the domain is defined as the source and sink terms in the advection equation. In the context of oil movement problem, the tracer corresponds to water and the non-traced fluid corresponds to oil (or hydrocarbons) in the reservoir while the source and sink locations correspond to injector and producer wells, respectively. Bear in mind that the oil flow problem can be more correctly modeled as a two-phase problem (water phase and oil phase). However for ease of execution and demonstration we chose single-phase flow problems where water injection is modeled as tracer flow that pushes the oil (non-traced fluid) in the reservoir, much like a contaminant in a fluid. Despite this approximation, the presented methodology is general enough and can be similarly applied to two-phase flow problems.

2.2.1 Problem description for robust optimal well control

We model the reservoir water injection process with an advection equation for tracer flow through porous media over the temporal domain $\mathcal{T} = [t_0, T] \subset \mathbb{R}$ and spatial domain $\mathcal{X} \subset \mathbb{R}^2$. The tracer flow models water flooding with the fractional variable $s(x, t) \in [0, 1]$, where $s(x, t)$ represents the fraction of water to oil at $x \in \mathcal{X}$ and $t \in \mathcal{T}$. The well controls $a(x', t)$ represent the source and sink terms in this equation, where $x' \in \mathcal{X}'$ (where $\mathcal{X}' \subset \mathcal{X}$) correspond to set of

well locations. Injector and producer flow controls are represented with a^+ (which constitutes of all the source locations in the domain and formulated as $\max(0, a)$) and a^- (which constitutes of all the sink locations in the domain and formulated as $\min(0, a)$), respectively. The optimal controls $a^*(x', t)$ are the solution of following closed-loop optimisation problem:

$$\max_{s(\cdot), a(\cdot)} \int_{t_0}^T \left(\sum_{x'} a^-(x', t)(1 - s(x', t)) \right) dt, \quad x' \in \mathcal{X}', t \in \mathcal{T} \quad (2.1a)$$

$$\frac{ds}{dt} = \frac{1}{\phi} (a^+ + sa^- - \nabla \cdot sv), \quad x \in \mathcal{X}, t \in \mathcal{T} \quad (2.1b)$$

$$s(\cdot, 0) = s_0, \quad v \cdot \mathbf{n} = 0, \quad (2.1c)$$

$$\sum_{x'} a^+(x', t) = - \sum_{x'} a^-(x', t) = c, \quad x' \in \mathcal{X}', t \in \mathcal{T} \quad (2.1d)$$

where, the objective function defined in Eq. (2.1a) represents the total oil flow out of the reservoir (oil production) and is maximized over the finite time interval \mathcal{T} . The integrand in this function is referred as Lagrangian term in control theory and is often denoted by $L(s, a)$. The water flow trajectory $s(x, t)$ is governed by an advection equation (Eq. (2.1b)) which is solved in couple with pressure equation $-\nabla \cdot (k/\mu)\nabla p = a$, where $p(x, t) \in \mathbb{R}$ is pressure. Porosity $\phi(x, \cdot)$, permeability $k(x, \cdot)$ and viscosity $\mu(x, \cdot)$ are the model parameters. Permeability k represents the model uncertainty and is treated as a random variable that follows a known probability density function \mathcal{K} with K as its domain. The initial and no flow boundary condition is defined in Eq. (2.1c), where \mathbf{n} denotes outward normal vector from the boundary of \mathcal{X} . Note that, the velocity v follows Darcy's law: $v = -(k/\mu)\nabla p$. Constraint defined in Eq. (2.1d) represents the fluid incompressibility assumption along with the fixed total source/sink term c which represents water injection rate in the reservoir. The controls $a(x', t)$ are discretized on time interval $t_0 < t_1 < \dots < t_m < t_{m+1} = T$.

2.2.2 RL framework for robust optimal well control

We propose to use model-free reinforcement learning algorithms to solve the optimal control problem defined in Eq. (2.1). A common approach in RL is to model the task as a Markov decision process. The process is defined as a quadruple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where $\mathcal{S} \subset \mathbb{R}^{n_s}$ is set of all possible states with dimension n_s , $\mathcal{A} \subset \mathbb{R}^{n_a}$ is a set of all possible actions with dimension n_a . State is represented with the tracer variable $s(x, t_m)$ and action with controls $a(x', t_m)$ such that it follows the constraint defined in Eq. (2.1d). $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is a transition function that follows *Markov property*. That is, it returns $s(x, t_{m+1})$ as a function of control $a(x', t_m)$ and state $s(x, t_m)$. Such transition function can be obtained by discretizing Eq. (2.1b) which returns $s(x, t_{m+1})$ by executing the controls $a(x', t_m)$ when in the state $s(x, t_m)$. The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ returns a real valued reward, $r(t_{m+1}) = \mathcal{R}(s(\cdot, t_m), a(\cdot, t_m), s(\cdot, t_{m+1}))$ for a particular transition between the states. The reward function for the problem (Eq. (2.1)) is

obtained by discretizing the objective function (Eq. (2.1a)) into control steps such that,

$$r(t_{m+1}) = \int_{t_m}^{t_{m+1}} L(s, a) dt. \quad (2.2)$$

The control policy, $\pi : \mathcal{S} \rightarrow \mathcal{A}$, defines an action $a(x', t_m)$ given the current state $s(x, t_m)$ (also written as $\pi(a|s)$).

The goal of reinforcement learning is to find an optimal policy $\pi^*(a|s)$ that maximizes expected discounted return, $\mathbf{G} = \sum_{m=1}^M \gamma^{m-1} r(t_m)$, where immediate rewards r are exponentially decayed by the discount rate $\gamma \in [0, 1]$ and M is the final control time step. Essentially, RL algorithms attempt to learn the optimal policy $\pi^*(a|s)$ from an initial policy, $\pi(a|s)$, by exploring state-action space with what is referred to as agent-environment interactions. Figure 2.1 shows a typical schematic of such agent-environment interaction. The term *agent* refers to the controller that follows the policy $\pi(a|s)$ while the *environment* consists of transition function, \mathcal{P} , and reward function, \mathcal{R} . The optimum solution can be obtained by following the policy $\pi^*(a|s)$

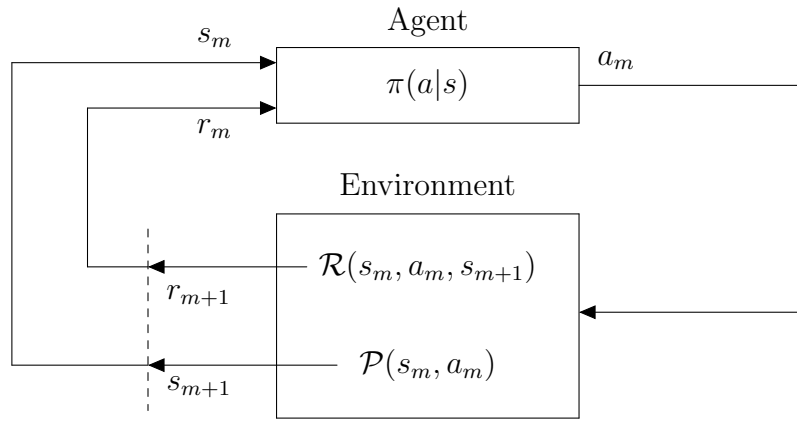


Figure 2.1: A typical agent-environment interaction in RL algorithms. state $s(x, t_m)$, action $a(x', t_m)$ and reward $r(t_m)$ are denoted with shorthand notations, s_m , a_m and r_m , respectively

throughout the complete control trajectory (also referred as an *episode* in RL literature) as shown in figure 2.2. Thus, optimal result (which refers to optimum oil recovery for optimal

$$\begin{array}{ccccccc}
 s_0 \rightarrow \mathcal{P}(s_0, a_0) & \longrightarrow & s_1 \rightarrow \mathcal{P}(s_1, a_1) & \longrightarrow & \dots & s_T \rightarrow \mathcal{P}(s_T, a_T) \\
 \downarrow & & \downarrow & & & \downarrow \\
 \pi^*(a|s) \longrightarrow a_0 & & \pi^*(a|s) \longrightarrow a_1 & & & \pi^*(a|s) \longrightarrow a_T
 \end{array}$$

Figure 2.2: optimal controls for complete control trajectory which refers to an episode in RL algorithms. state $s(x, t_m)$ and action $a(x', t_m)$ are denoted with shorthand notations, s_m and a_m , respectively

well control problem), $\mathbf{R}^{\pi^*(a|s)}$, is obtained by adding the reward $r(\cdot)$ at each time-step of such

episode:

$$\mathbf{R}^{\pi^*(a|s)} = \sum_{m=0}^{M-1} \left[\int_{t_m}^{t_{m+1}} L(s, \pi^*(a|s)) dt \right]. \quad (2.3)$$

Note that the optimal result, which is used in the rest of the paper to evaluate the policy, is not exponentially decayed with the discount factor γ . The policy $\pi^*(a|s)$ is said to be *robust* if it is able to achieve optimal results for a stochastic environment controlled by the uncertain parameter k , defining the permeability in the Darcy flow equation.

If we treat the parameter k as a deterministic fixed value, policy learning is fairly straightforward. The policy learned in such a way is termed as *frozen* policy (a term used by [30]). In this study we aim to find a robust policy that accounts for the variability in k . To the best of our knowledge, so far, such policy is learned by simply incurring samples from the distribution \mathcal{K} , in each episode of the learning process (also known as static domain randomisation method [46]). Such policy can be robust enough if the samples used in the learning process cover most of its domain K (In other words, when K is well explored). For this reason, robust optimal policy learning naturally requires higher number of agent-environment interactions as compared to that in learning frozen policy. This could be problematic if each agent-environment interaction (solving the governing Equations (2.1b), for instance) is computationally intensive, which is common in most simulation-based optimal control problems like optimal well control. Furthermore, samples incurred in this learning process often tend to be from the high probability region of the distribution domain causing the policy to be biased towards them.

robustness of frozen policy: We denote frozen policy learned by keeping the parameter k fixed as $\pi^*(a|s; k)$. Lets define a distance function $D : K \times K \rightarrow \mathbb{R}$ which returns the distance between k and k' as $D(k, k')$. Naturally, the frozen policy can be applied to the simulation (or environment) that uses k' as the parameter instead of k (denoted as $\pi^*(a|s; k \Rightarrow k')$). Due to continuous nature of governing Equations (2.1b), effectiveness of the policy, $\pi^*(a|s; k \Rightarrow k')$, is inversely related to the distance $D(k, k')$. We can define an acceptable near-optimal solution limit obtained with $\pi^*(a|s; k \Rightarrow k')$ when k' is in the neighborhood (δ) of k . In other words, we can say the policy $\pi^*(a|s; k \Rightarrow k')$ can be considered robust when $D(k, k') < \delta$.

This argument can be extended for multiple parameters: k_1, k_2, \dots, k_l . In this case, we learn the policy, $\pi^*(a|s; \mathbf{k})$, by randomly choosing any one of the parameter value from the vector $\mathbf{k} = (k_1, k_2, \dots, k_l)$ at every episode in the learning process. Subsequently, the policy, $\pi^*(a|s; \mathbf{k} \Rightarrow \mathbf{k}')$, can yield a near optimal value for the vector $\mathbf{k}' = (k'_1, k'_2, \dots, k'_l)$, given that $\min_i D(k_i, k'_i) < \delta, \forall j \in \{1, 2, \dots, l\}$. If the vector \mathbf{k} is chosen such that the union of neighbourhood of all its values $k_i, \forall i \in \{1, 2, \dots, l\}$, cover the domain K , the policy $\pi^*(a|s; \mathbf{k})$ yields at least near optimal solution for any sample $k' \sim \mathcal{K}$. Figure 2.3 shows an example of such choice of \mathbf{k} values for the uncertainty distribution \mathcal{K} . For such a *well spread* choice of \mathbf{k} , the policy $\pi^*(a|s; \mathbf{k})$ can be said to be robust under the uncertainty distribution \mathcal{K} .

Implementation: Although the above-mentioned policy $\pi^*(a|s)$ provides optimal solution for the problem defined in Equation (2.1), it is not applicable for systems with partially observable

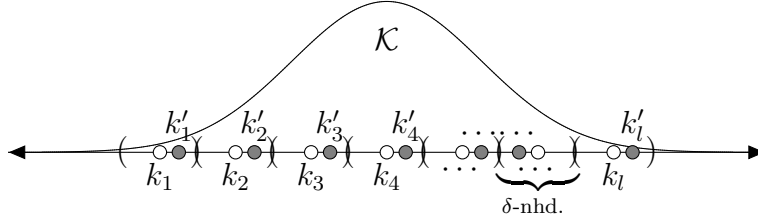


Figure 2.3: a wells spread choice of samples for some uncertainty distribution \mathcal{K} (depicted as one-dimensional for illustration purpose) to learn the robust optimal policy $\pi^*(a|s; \mathbf{k})$. Members of \mathbf{k} are colored in white while the members of \mathbf{k}' are shown in grey.

state space. For instance, in optimal well control problems reservoir information is generally only available at well locations. As a result we provide the agent with the available *observation* $o(x', t_m)$ as its state. For this study, observation $o(x', t_m)$ is represented with a set of saturation and pressure values at well locations x' and time t_m . Note that, with such representation of states, we break the underlying assumption of Markov property of the transition function. Here, we assume that transition between the observations approximately follow the Markov property.

We choose l well spread samples of uncertainty distribution to learn a robust policy. This is achieved with a clustering analysis (using k-means clustering method) of the domain K . The *training vector* \mathbf{k} is constructed with samples of \mathcal{K} which are located at the cluster centers. The policy, $\pi^*(a|s; \mathbf{k})$, is learned by randomly selecting the parameter k from the training vector $\mathbf{k} = \{k_1, k_2, \dots, k_l\}$. Average *training return* is calculated by averaging the returns of policy $\pi(a|s; \mathbf{k})$ on l simulations, each with a separate parameter k from \mathbf{k} .

$$\mathbf{R}^{\pi(a|s; \mathbf{k} \Rightarrow \mathbf{k})} = \frac{1}{l} \sum_{i=1}^l \left[\sum_{m=0}^{M-1} \left(\int_{t_m}^{t_{m+1}} L(s, \pi(a|s; \mathbf{k} \Rightarrow k_i)) dt \right) \right] \quad (2.4)$$

The robustness of this policy is assessed by applying it on l simulations, each with a new unseen sample, $k' \sim \mathcal{K}$, as its parameter. The samples for evaluation are chosen randomly from each cluster. The *evaluation vector* $\mathbf{k}' = \{k'_1, k'_2, \dots, k'_l\}$ represents the set of such samples. Robustness of the policy $\pi(a|s; \mathbf{k})$ is evaluated by monitoring the average *evaluation return* $\mathbf{R}^{\pi(a|s; \mathbf{k} \Rightarrow \mathbf{k}')}$:

$$\mathbf{R}^{\pi(a|s; \mathbf{k} \Rightarrow \mathbf{k}')} = \frac{1}{l} \sum_{i=1}^l \left[\sum_{m=0}^{M-1} \left(\int_{t_m}^{t_{m+1}} L(s, \pi(a|s; \mathbf{k} \Rightarrow k'_i)) dt \right) \right]. \quad (2.5)$$

In a nutshell, we expect to train the policy $\pi(a|s; \mathbf{k})$ such that $\mathbf{R}^{\pi(a|s; \mathbf{k} \Rightarrow \mathbf{k}')}$ is maximized.

We employ PPO and A2C algorithms to solve this problem. The RL algorithm parameters are tuned in order to maximize the average returns, $\mathbf{R}^{\pi(a|s; \mathbf{k} \Rightarrow \mathbf{k})}$. As the values of training vector, \mathbf{k} , fairly represent the variety of the domain K , we expect convergence of average *evaluation* return value towards $\mathbf{R}^{\pi^*(a|s; \mathbf{k} \Rightarrow \mathbf{k}')}$ by the end of the learning process. The baseline optimal results are computed using differential evolution (DE) algorithm [47]. DE algorithm is used to solve l optimisation problems as described in Equations (2.1) each with the parameter k

replaced by a fixed value from evaluation vector \mathbf{k}' . The reference value for $\mathbf{R}^{\pi^*(a|s;\mathbf{k}\Rightarrow\mathbf{k}')}$ is taken as average of these l solved optimal objective functions. Both the RL algorithms are tuned such that $\mathbf{R}^{\pi(a|s;\mathbf{k}\Rightarrow\mathbf{k}')}$ converges in the range of this reference value.

2.2.3 RL algorithms

We choose model-free RL algorithms to avoid any effect of model learning on policy learning. Two state-of-the-art policy-based algorithms (A2C and PPO) are used to solve the optimal control problem under consideration.

2.2.3.1 Advantage actor-critic algorithm

A2C [38] is a policy gradient algorithm that models the stochastic policy, $\pi_\theta(s|a)$, with a neural network. Essentially, the network parameters θ are obtained by optimizing for the objective function,

$$J_{a2c}(\theta) = \hat{\mathbb{E}}_t \left[\log \pi_\theta(a_t|s_t) \hat{A}_t \right], \quad (2.6)$$

where, \hat{A}_t is an estimator for advantage function at timestep t and the term, $\hat{\mathbb{E}}_t[\cdot \cdot \cdot]$, is empirical average over finite batch of samples collected through agent-environment interactions. Gradient estimator of policy network, $\nabla_\theta J(\theta)$, is obtained by differentiating Eq. (2.6) which is done with automatic differentiation algorithm. The advantage function estimator, \hat{A}_t , is computed using generalized advantage estimator [48] which is derived from the value function V_t . The value function estimator \hat{V}_t is learned through a separate neural network termed as the *critic* network. Definitions of advantage and value functions are described in A.1. Algorithm 1 illustrates a broad outline for implementation of A2C algorithm in this study. In order to reduce computational time, the iterative data sampling for objective function is performed in parallel on N processors followed by a synchronous gradient update. Note that in every policy iteration in total T control steps are run where environment is reset with a new permeability sample from the \mathbf{k} after every terminal step of the episode.

2.2.3.2 Proximal policy optimisation algorithm

If the gradient step in A2C is too large, the policy may astray which in turn will produce bad samples causing divergence in the solution. As a result, we have to select very small step size which slows the learning process. Schulman et al. [39] introduced PPO algorithm that make sure the gradient steps are small enough to make the algorithm data efficient. This is done by formulating the network objective function in terms of a ratio of two policies (old and new) using principle of importance sampling. Appropriate gradient steps are chosen by clipping the ratio of old and new policy within the range $[1 - \epsilon, 1 + \epsilon]$, where ϵ is generally a small fractional

¹In practice, a single integrated neural network is used for both, actor and critic networks. As a result, objective function for automatic differentiation is summation of Eq. (2.6) and value loss function for critic network. Please refer Algorithm S2 from [38] for the formulation of value loss function.

Algorithm 1 Policy Robustness Evaluation using A2C

-
- 1: **Input:** Number of actors N , and number of steps in each policy iteration T
 - 2: Obtain training vector \mathbf{k} , and evaluation vector \mathbf{k}' using cluster analysis of the predefined uncertainty distribution \mathcal{K}
 - 3: **for** $iteration = 1, 2, \dots$ **do**
 - 4: **for** $actor = 1, 2, \dots, N$ **do**
 - 5: Run policy π_θ in environment for T time steps (which corresponds to in total E episodes where the environment permeability is set to a sample from training vector \mathbf{k} , at the beginning of every episode)
 - 6: Compute value function estimates $\hat{V}_1, \dots, \hat{V}_T$ using critic network
 - 7: Compute advantage function estimates $\hat{A}_1, \dots, \hat{A}_T$
 - 8: **end for**
 - 9: Optimize $J_{a2c}(\theta)$ ¹(Eq. (2.6)), with single epoch and batch size NT
 - 10: $\theta_{old} \leftarrow \theta$
 - 11: compute and record training return $\mathbf{R}^{\pi(a|s;\mathbf{k}\Rightarrow\mathbf{k})}$, and evaluation return $\mathbf{R}^{\pi(a|s;\mathbf{k}\Rightarrow\mathbf{k}')}$
 - 12: **end for**
-

number. The modified objective function for policy network is defined as,

$$J_{ppo}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t(s_t, a_t), \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t(s_t, a_t)) \right], \quad (2.7)$$

where $r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$ and $\pi_{\theta_{old}}(a_t|s_t)$ is old policy. Algorithm 2 illustrates the implementation of PPO algorithm in the presented methodology.

Algorithm 2 Policy Robustness Evaluation using PPO

-
- 1: **Input:** Number of actors N , number of steps in each policy iteration T , number of epochs K and minibatch size M
 - 2: Obtain training vector \mathbf{k} , and evaluation vector \mathbf{k}' using cluster analysis of the predefined uncertainty distribution \mathcal{K}
 - 3: **for** $iteration = 1, 2, \dots$ **do**
 - 4: **for** $actor = 1, 2, \dots, N$ **do**
 - 5: Run policy π_θ in environment for T time steps (which corresponds to in total E episodes where the environment permeability is set to a sample from training vector \mathbf{k} , at the beginning of every episode)
 - 6: Compute value function estimates $\hat{V}_1, \dots, \hat{V}_T$ using critic network
 - 7: Compute advantage function estimates $\hat{A}_1, \dots, \hat{A}_T$
 - 8: **end for**
 - 9: Optimize $J_{ppo}(\theta)$ ²(Eq. (2.7)), with K epochs and minibatch size $M \leq NT$
 - 10: $\theta_{old} \leftarrow \theta$
 - 11: compute and record training return $\mathbf{R}^{\pi(a|s;\mathbf{k}\Rightarrow\mathbf{k})}$, and evaluation return $\mathbf{R}^{\pi(a|s;\mathbf{k}\Rightarrow\mathbf{k}')}$
 - 12: **end for**
-

²The objective function for integrated actor-critic network, in the PPO implementation, is the summation of actor loss term (Eq. (2.7)), value loss term and entropy loss term. Readers are referred to [39] for the detailed definition of value loss term and entropy loss term.

2.2.4 Differential evolution algorithm

We employ DE algorithm [47] as a baseline for assessing the results obtained using PPO and A2C. Essentially, DE is a population based stochastic optimisation algorithm which employs evolutionary ideas like crossover and mutation to find optimal arguments. In order to solve the optimisation problem (Eq. (2.1)) using DE, we represent the argument with a set of controls $\mathbf{a} = \{a(x', t_0), \dots, a(x', T)\}$ such that it follows the constraints defined in Eq. (2.1d). The *fitness* of such argument is computed with the objective function (Eq. (2.1a)) by solving the governing Eq. (2.1b). DE algorithm initiates its argument search with a set \mathbf{A} of random arguments which is referred as *population* (i.e. $\mathbf{A} = \{\mathbf{a}_1, \dots, \mathbf{a}_p\}$). Using a crossover criteria, certain arguments (say, i^{th} argument in \mathbf{A} : \mathbf{a}_i) are evolved as ,

$$\mathbf{a}'_i = \mathbf{a}^* + F(\mathbf{a}_{r1} + \mathbf{a}_{r2}),$$

where \mathbf{a}'_i is updated value for \mathbf{a}_i , \mathbf{a}^* is the best argument (i.e. the one corresponding to maximum fitness) in the population so far, $F \in [0, 2]$ is mutation parameter, \mathbf{a}_{r1} and \mathbf{a}_{r2} are randomly selected arguments from the population. \mathbf{a}_i is replaced with \mathbf{a}'_i if the fitness of \mathbf{a}'_i is higher than that for \mathbf{a}_i . The optimum solution is obtained by repeating such evolution for a number of iterations or until a certain convergence criteria is met.

Note that DE algorithm's parameter search space is wider than that for RL. This is because the optimum parameters do not have to follow a mapping $\pi(a|s)$. For this reason, we expect DE algorithm to achieve more optimal controls as compared to RL algorithms due to its potential to achieve global optima.

2.2.5 K-means clustering

We employ connectivity distance [49] measure in order to represent the variation in dynamical response of permeability samples. The connectivity distance matrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ for a large number (N) of samples of \mathcal{K} is written by,

$$\mathbf{D}(k_i, k_j) = \sum_{x''} \int_{t_0}^T [s(x'', t; k_i) - s(x'', t; k_j)]^2 dt, \quad (2.8)$$

where, x'' are a set of spatial locations in the domain \mathcal{X} and $s(x'', t; k_i)$ refers to solution of governing Eq. (2.1b) with the uncertainty parameter k_i when all control wells are kept equally open. In order to be able to visualize the connectivity dissimilarity among samples of \mathcal{K} , we employ multi-dimensional scaling on the distance matrix \mathbf{D} to obtain a set of N two-dimensional coordinates represented with d_1, d_2, \dots, d_N . In other words, coordinates d_1, d_2, \dots, d_N correspond to samples k_1, k_2, \dots, k_N of \mathcal{K} such that it represents connectivity distance measure defined in Eq. (2.8) among its values. The coordinates d_1, d_2, \dots, d_N are divided in l sets S_1, S_2, \dots, S_l

obtained by solving the optimisation problem:

$$\arg \min_S \sum_i^l \sum_{d_j \in S_i} \|d_j - \mu_{S_i}\|,$$

where μ_{S_i} is average of all coordinates in the set S_i . The training vector \mathbf{k} is a set of l samples of \mathcal{K} where each of its value k_i correspond to the one nearest to μ_{S_i} .

2.3 Numerical experiments

We evaluate the effectiveness of RL in solving robust optimal well control problems using two test cases representing two distinct permeability uncertainty distributions. Numerical solutions of the governing equations are obtained by using finite volume discretization. The pressure equation is discretized using two point flux approximation method while the saturation equation is discretized using implicit upwind scheme. Readers are referred to [50] for more details on numerical methodology. For both cases, the values for model parameters emulate those in the benchmark reservoir simulation case, SPE-10 model 2 [51].

2.3.1 model parameters

Reservoir simulation parameters for both the cases, corresponding to Eq. (2.1), are delineated in table 2.1. The permeability k is treated as the uncertain parameter with its unit as milliDarcy. As per the convention in geostatistics, we assume that the distribution of $\log(k)$ is known and is denoted by \mathcal{G} . As a result, we treat $g = \log(k)$ as a random variable in the problem description defined in Eq. (2.1). Uncertainty distributions for test cases 1 and 2 are denoted with \mathcal{G}_1 and \mathcal{G}_2 , respectively.

Table 2.1: Reservoir model parameters

| | Test case 1 | Test case 2 | units |
|--------------------------------|-------------|-------------|----------------------|
| spatial domain \mathcal{X} | (1200×1200) | (1200×1200) | ft ² |
| temporal domain \mathcal{T} | [0,125] | [0,25] | days |
| initial saturation s_0 | 0.0 | 0.0 | – |
| viscosity μ | 0.3 | 0.3 | cP |
| porosity ϕ | 0.2 | 0.2 | – |
| number of producers n_p | 31 | 4 | – |
| number of injectors n_i | 31 | 1 | – |
| total injector flow $\sum a^+$ | 2304 | 8064 | ft ² /day |

Test case 1 (Channel like permeability distribution): Figure 2.4a shows schematic of the first test case (inspired from the case study done by [44]). A total number of 31 injectors are placed on the left edge of the domain while an equal number of producers are placed symmetrically on the right side. A linear high permeability channel connects from left to right side of the domain. The channel location is parameterized with its left and right distance (l_1 and l_2)

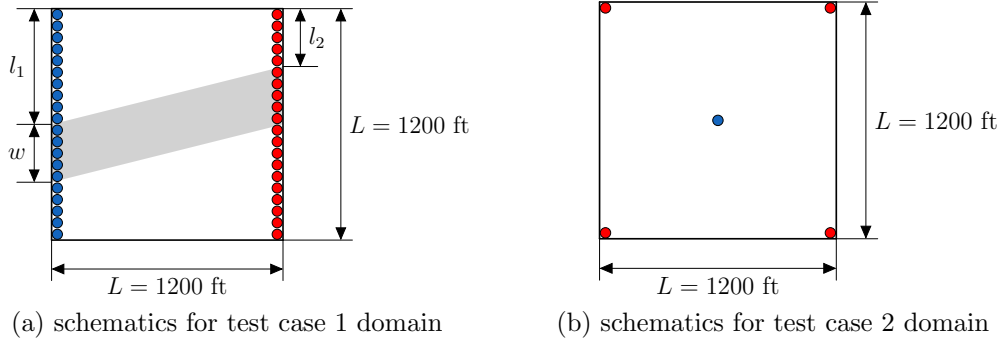


Figure 2.4: the producers and injectors are highlighted with red and blue colors, respectively. parameters $(w, l_1$ and $l_2)$ for test case 1 log permeability are shown in fig (a), where high permeability channel is colored in gray.

from the top and channel width w . These parameters follow uniform distributions defined as, $w \sim U(120, 360)$, $l_1 \sim U(0, L - w)$ and $l_2 \sim U(0, L - w)$, where L is domain length. log permeability g is sampled from the uncertainty distribution \mathcal{G}_1 :

$$g \sim \mathcal{G}_1(w, l_1, l_2).$$

To be specific, log permeability g at a location (x, y) is formulated as:

$$g(x, y) = \begin{cases} g_{\text{high}} & \text{if } \frac{l_2 - l_1}{L}x + l_1 \leq y \leq \frac{l_2 - l_1}{L}x + l_1 + w, \\ g_{\text{low}} & \text{otherwise,} \end{cases}$$

where x and y are horizontal and vertical distances from the upper left corner of the domain illustrated in figure 2.4a. The values for g_{high} and g_{low} (5.5 and -2, respectively) are inspired from *Upperness* permeability distribution specified in SPE-10 model 2 case (refer to A.2 for details). Figure 2.6a illustrate various samples drawn from the distribution \mathcal{G}_1 .

Test case 2 (Spatially correlated smooth permeability distribution): We use test case 2 to represent uncertainty distribution of a smoother permeability field. Figure 2.4b illustrates reservoir domain for this case. It comprises of four producers located at four corners of the domain and an injector located at the center of the domain. The permeability distribution for this case is considered as a log normal distribution which is constrained with fixed values at well locations. As a result, log permeability g is sampled from the normal distribution \mathcal{G}_2 :

$$g \sim \mathcal{G}_2(\mu_g, \Sigma_g), \text{ where,} \quad (2.9)$$

$$\mu_g = 2.41, \quad (2.10)$$

$$\Sigma_g = C(x, x) - C(x, x')C(x', x')^{-1}C(x, x'), \quad (2.11)$$

where, $C(x, x')$ is the co-variance matrix between unconstrained domain locations x and con-

strained locations x' while μ_g correspond to the constrained log-permeability value at the well locations. The co-variance matrix is calculated using an exponential kernel as:

$$C(a, b) = \sigma^2 \exp -\frac{\|a - b\|}{l}. \quad (2.12)$$

We choose correlation length l as 240 ft (20% of domain length) and the variance amplitude σ as 2.5. The values μ_g , σ and l were chosen to fit permeability distribution in the same range of that in *Tarbert* case specified in SPE-10 model 2 case (refer A.2). Examples of samples drawn from the distribution \mathcal{G}_2 are shown in figure 2.6b.

2.3.2 RL problem formulation:

Both, PPO and A2C, algorithms attempt to learn neural network parameters θ to learn the policy $\pi_\theta(a|s)$. We choose five step episode which is obtained by dividing the temporal domain \mathcal{T} into five control steps. The optimisation potential of the problem can be improved if we choose a higher than 5 control steps. However, we choose 5 control steps for the ease of execution and demonstration. The episode steps are denoted with t_m where $m \in \{1, 2, \dots, 5\}$. State is represented by observation $o(x', t_m)$ which is a vector of saturation and pressure values at all well locations. However, since the saturation at injectors is always constant (one), we don't include it in the observation. As a result, the observation vector is of the size $2n_p + n_i$ (i.e. 93 for test case 1 and 9 for test case 2) which forms the input of the policy network $\pi_\theta(a|s)$. Action $a(x', t_m)$ is represented with a vector the size of number of control wells $n_p + n_i$ (i.e. 62 for test case 1 and 5 for test case 2). In order to maintain constraint defined in Eq. (2.1d), we represent the action vector with weights, w_i s, such that $0.001 \leq w_i \leq 1$ (i.e. action vector is written as, $a(x', t_m) = (w_1, \dots, w_{n_p}, w_{n_p+1}, \dots, w_{n_p+n_i})$). Using these weights, flow through i th producer, $a^-(x'_i, \cdot)$, is computed as,

$$a^-(x'_i, \cdot) = -\frac{w_i}{\sum_{i=1}^{n_p} w_i} c.$$

Similarly, flow through i th injector, $a^+(x'_i, \cdot)$, is written as,

$$a^+(x'_i, \cdot) = \frac{w_{i+n_p}}{\sum_{i=1}^{n_i} w_{i+n_p}} c.$$

The reward function defined in Eq. (2.2) is normalized by dividing it with total pore volume ($\phi \times lx \times ly$) in order to obtain the reward in the range $[0,1]$. The normalized reward represents *recovery factor* or *sweep efficiency* for oil movement problem in petroleum reservoir.

We use clustering strategy explained in section 2.2.5 where we choose total number of samples, N , and clusters, l , to be 1000 and 16 for both uncertainty distributions, \mathcal{G}_1 and \mathcal{G}_2 . Training vector \mathbf{k} is obtained with samples k_1, \dots, k_{16} each corresponding to a cluster center. Figure 2.5a and 2.5b show cluster plots for samples drawn from \mathcal{G}_1 and \mathcal{G}_2 permeability distributions, respectively. Permeability samples for test case 1 are distributed in the shape of an acute angle

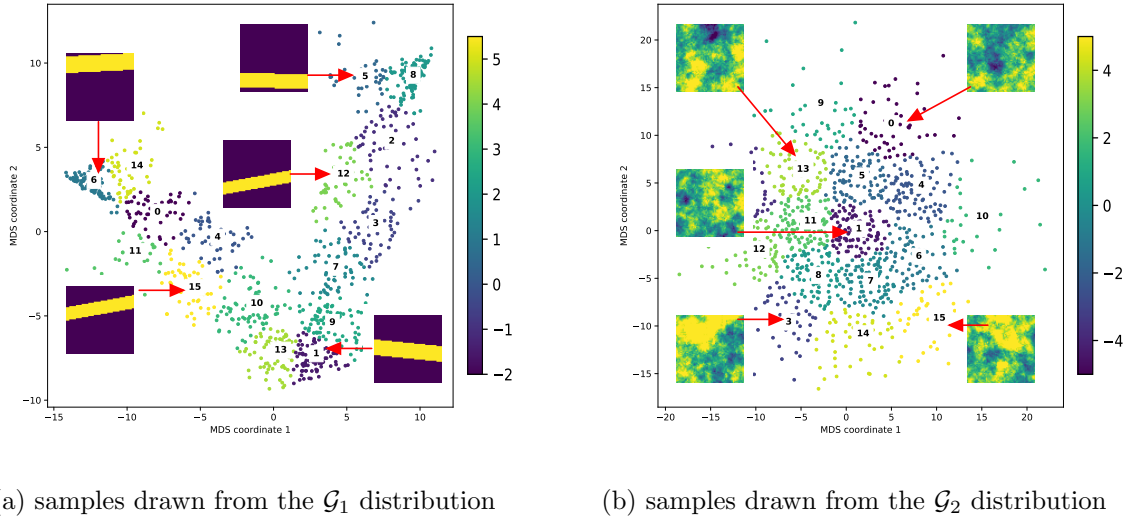


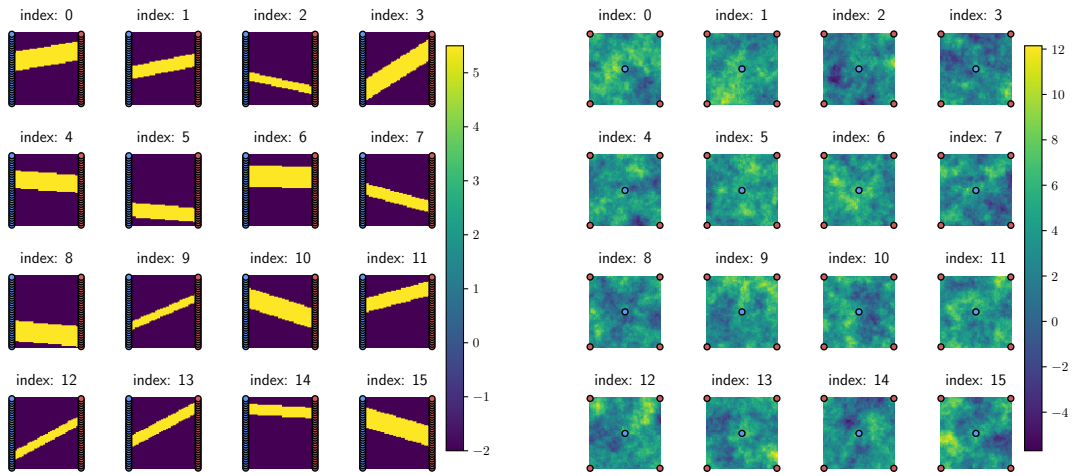
Figure 2.5: clustering of log-permeability fields (unit: mD) for test cases 1 and 2

where samples in the vertex region correspond to high permeability channel at the central region, samples on the left arm correspond to high permeability channel in the upper region while samples in the right arm correspond to high permeability channel in the lower region of the domain. For test case 2, samples with more or less axisymmetric high permeability region are located in the central area in figure 2.5b (e.g., cluster 1). The samples corresponding to eccentric high permeability regions are located outside as shown with examples \mathbf{k}_{13} (lower left region), \mathbf{k}_3 (upper left region), \mathbf{k}_{15} (upper right region) and \mathbf{k}_0 (lower right region) in figure 2.5b. In order to represent well spread domain of \mathcal{G}_1 and \mathcal{G}_2 distributions, the 16 samples, each randomly chosen from a cluster, forms the evaluation vector \mathbf{k}' . These evaluation samples are shown in figure 2.6a and 2.6b for test cases 1 and 2, respectively.

Figure 2.7 outlines the general schematics for agent-environment interactions in PPO and A2C algorithms for robust optimal control problem. Since these algorithms are stochastic in nature, we provide training and evaluation returns as a mean corresponding to three distinct seed values. These results are benchmarked against DE algorithm optimisation results. Parameters used for all algorithms along with the confidence range of learning plots are presented in A.3.

2.4 Results and discussion

We refer to the control policy in which all wells are equally open as the *base* policy. When the first test case is operated with base policy, most of the water flooding take place in the high permeability channel causing poor sweep efficiency in the low permeability region. Naturally, the optimal policy is to restrict the flow through wells which are in the region nearby high permeability channel. Using DE algorithm, we obtain 16 optimized solutions for optimal control problem defined in Equations (2.1) where each value in evaluation vector \mathbf{k}' is treated as fixed permeability k . These results act as a reference to optimal solutions obtained using PPO



(a) samples of \mathcal{G}_1 in evaluation vector \mathbf{k}'

(b) samples of \mathcal{G}_2 in evaluation vector \mathbf{k}'

Figure 2.6: log-permeability plots (unit: mD) for evaluation data for test cases 1 and 2

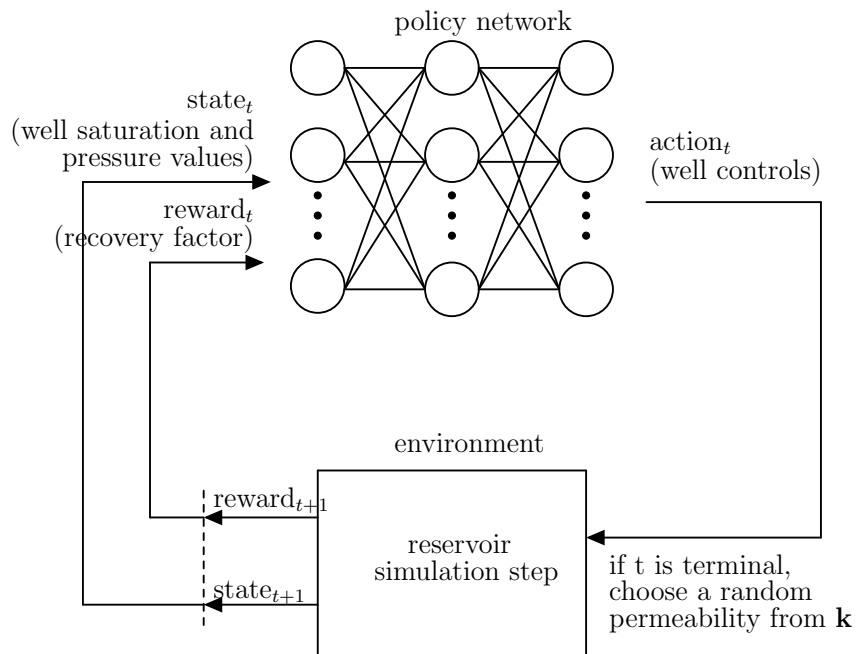


Figure 2.7: RL algorithm agent-environment interaction schematics to learn robust optimal well control policy

and A2C algorithms. Reference value for mean evaluation return $\mathbf{R}^{\pi^*(a|s;\mathbf{k}\Rightarrow\mathbf{k}')}$ is obtained by averaging DE results of these 16 problems. Note that DE algorithm is not a suitable method to solve the robust optimal control problem since it can provide optimal controls only for certain permeability samples as opposed to PPO or A2C algorithms where we try to learn the policy that is applicable to all samples of permeability distribution. However, DE results are used as the reference since they provide the upper bounds achieved by direct optimization on sample by sample basis. Figure 2.8 shows plots for training ($\mathbf{R}^{\pi(a|s;\mathbf{k}\Rightarrow\mathbf{k})}$) and evaluation ($\mathbf{R}^{\pi(a|s;\mathbf{k}\Rightarrow\mathbf{k}')$) returns versus total number of episodes for PPO and A2C learning process. As can be seen, PPO and A2C algorithms successfully learned the robust optimal policy and their average evaluation returns $\mathbf{R}^{\pi^*(a|s;\mathbf{k}\Rightarrow\mathbf{k}')$ are within the range of DE results. We also present results for a frozen PPO policy trained using a fixed permeability located at index 1 in the training vector \mathbf{k} (indicated with dotted green line in figure 2.8). We note that the training vector \mathbf{k} for frozen PPO case only comprises of a single permeability realization. This frozen policy is not robust as it performs poorly on unseen permeabilities as demonstrated when we plot $\mathbf{R}^{\pi(a|s;\mathbf{k}\Rightarrow\mathbf{k}')$ value in its learning process. Furthermore, learning plot for PPO algorithm with full state representation is illustrated with red dotted line. In this case, we provide the agent with saturation values at each grid point in the domain (i.e. with a vector of length $61 \times 61 = 3721$). Policy learning with full state information are in the same range of that with only well observation state representation. In other words, information of well observations is enough to form optimal policy for this case.

Figure 2.9 plots optimum recovery factors (in %) corresponding to each evaluation permeability in the vector \mathbf{k}' . Results of PPO and A2C policy are comparable to the DE results which are independently optimized for each permeability field. Figure 2.10 illustrates the optimum controls corresponding to evaluation permeability at third and fifth indices of \mathbf{k}' . The controls for injectors and producers are shown in blue and red colored circles, respectively. Note that the radius of the circle at certain well location is proportional to flow through that well. That is, the radius of the circle at certain well location is proportional to the flow control opening of the corresponding well. As can be seen in figure 2.10, the optimal control policy to restrict flow controls in the high permeability region is successfully learned using PPO and A2C.

In the second test case, all reservoir parameters and well locations are axisymmetric except the permeability field. As a result, there is an imbalance in the flow direction from the central injector. The optimal flow control policy is to govern the well controls so that balanced amount of sweeping can be maintained in all four quadrants of the reservoir. For instance, if water sweeps uniformly in all quadrants of reservoir except the upper left, the optimal policy should increase the flow through upper left producer while restricting the flow through rest of the producers (i.e. govern the controls to cope with the imbalance in the upper left quadrant). For the five-spot case under consideration, the optimal policy has in total 10 modes: four due to imbalance in single quadrant and six due to imbalance in a pair of quadrants.

Note that these 10 modes represent the full extent of uncertainty in the optimal policy for the

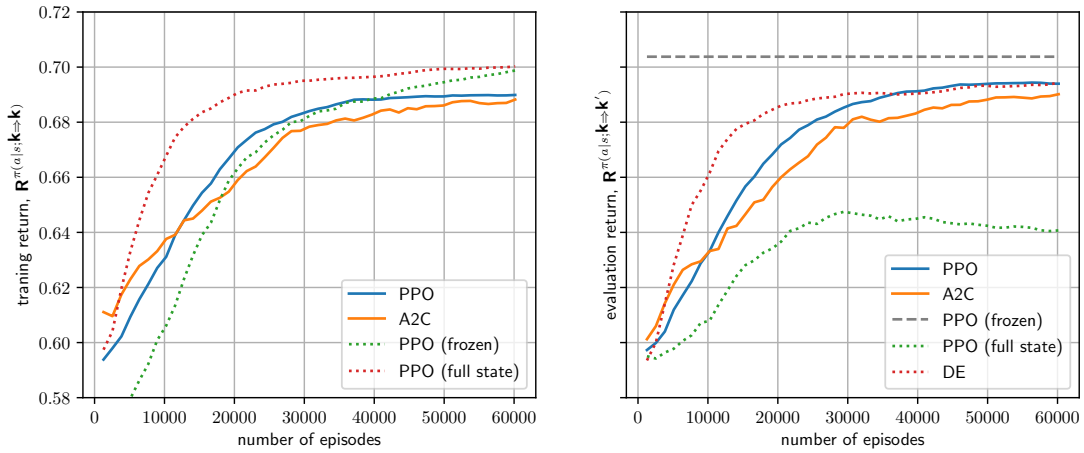


Figure 2.8: Test case 1: monitoring plots for average training return $R^{\pi(a|s; k \Rightarrow k)}$ (on left) and evaluation return $R^{\pi(a|s; k \Rightarrow k')}$ (on right) for learning process in PPO, A2C and frozen PPO. The evaluation return value is compared with the optimisation results obtained using DE

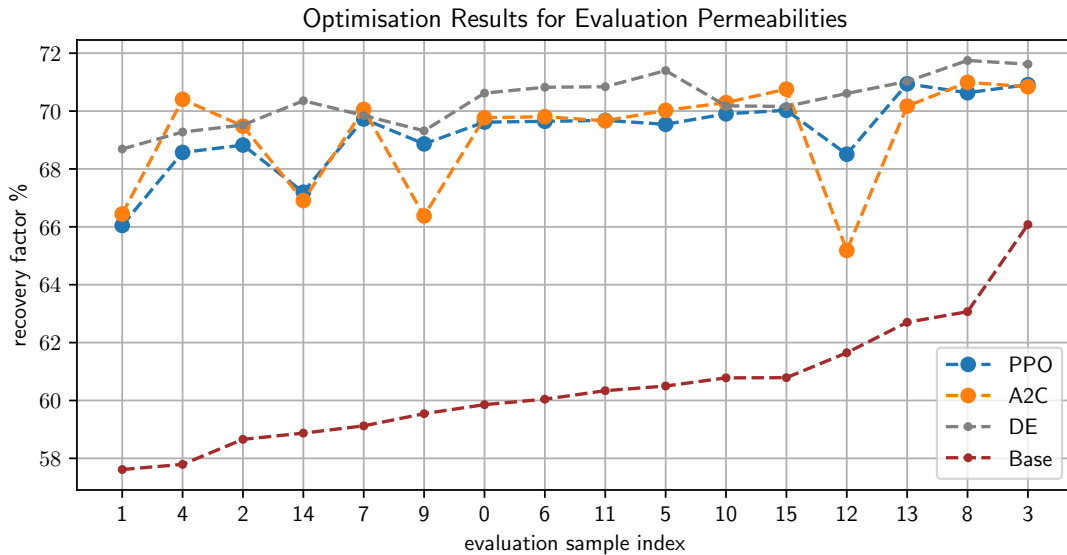
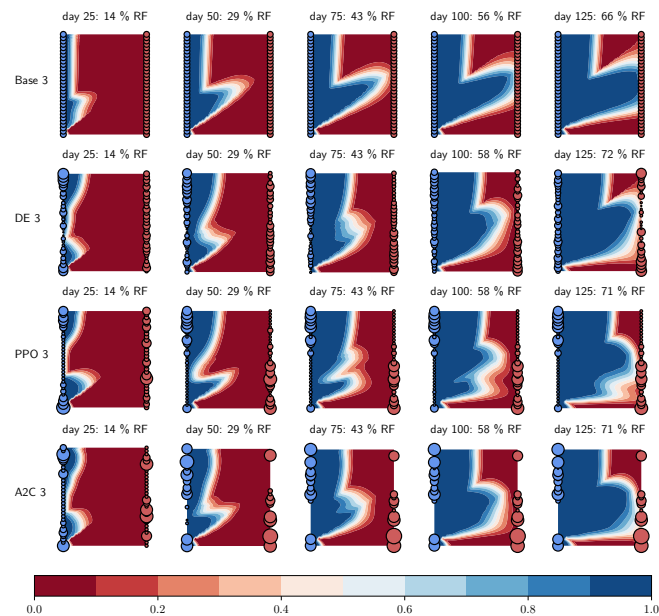
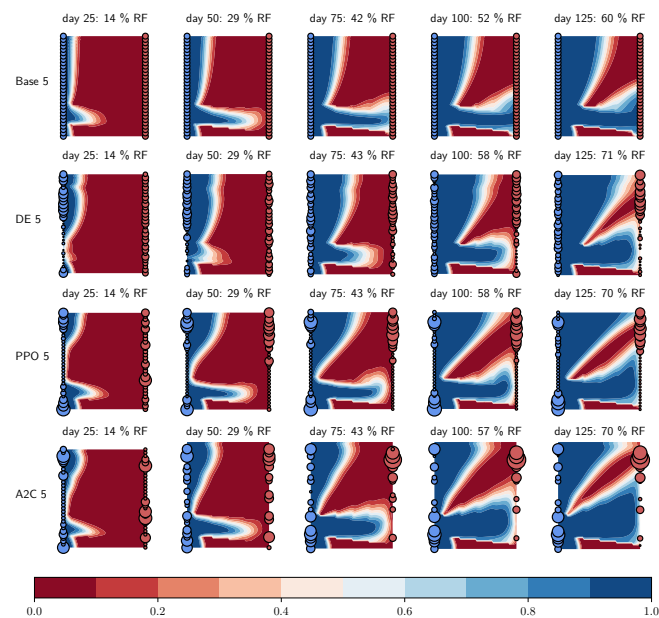


Figure 2.9: Test case 1: comparison of optimum recovery factor (in %) for each permeability value from evaluation vector k' . Results of of PPO, A2C and DE are compares with base control actions (all control valves equally open) results



(a) Evaluation permeability index 3



(b) Evaluation permeability index 5

Figure 2.10: Test case 1: optimal well controls for permeability values k'_3 and k'_5 (illustrated with saturation contour plots). Producer and injector flow controls are indicated with red and blue circles, respectively. Values of flow controls are proportional to the circle radius.

second test case. For the first test case, the uncertainty in the optimal policy is subject to the location of the high permeability channel in the domain. This can be divided in 9 major groups depending on the location of the channel. For instance, we can assume the channel to start and end at either the upper, lower, or middle part of the domain (i.e. $3 \times 3 = 9$ combinations). As a result, we wanted to choose the number of clusters in the permeability uncertainty distribution to be more than 9 for test case 1 and 10 for test case 2. In order to maintain the uniformity presented case studies we demonstrate the cluster-based domain randomization methodology in total 16 numbers of clusters.

confusion in RL policy learning: Policy $\pi_{\theta}(a|s)$ is learned through numerous agent-environment interactions experienced with 16 permeability field instances in training vector \mathbf{k} . By definition, the optimal policy returns the action a that corresponds to maximum return episode from current state s . Since the first state s_0 is same for all permeability fields (according to initial condition defined in Eq. (2.1c)), the first action $\pi_{\theta}(a_0|s_0)$ is always the one that correspond to a certain permeability field in training vector \mathbf{k} which produces maximum total return. So if we imagine \mathbf{k}_5 to be such a permeability which happens to follow one of the ten optimal policy modes (say mode 7). The first action a_1 will always be the one that correspond to mode 7 policy. This is obviously undesirable when we apply this policy on permeability fields which correspond to another mode of optimal policy. In order to avoid this *confusion in policy learning*, we train RL policies from second step onward. Since the second step of the episode is different for different permeability fields, RL policy learning doesn't face this confusion anymore. By default we treat the first action to follow base policy (i.e. all wells open equally). Note that the first action for test case 1 RL optimal policies is also identical for all cases (refer to figure 2.10). However, since the optimal policy's nature is not modal, the resulting sub-optimality is not as much prominent for this case.

Figure 2.11 shows learning plots for training and evaluation returns for PPO, A2C and DE algorithms. Similar to test case 1 results, PPO and A2C algorithms successfully learn robust optimal policy. RL policies learned with well observations show results in the same range with the PPO policy with full state representation. As expected, the frozen policy's lack of robustness can be seen in evaluation return learning plot. PPO, A2C and DE optimisation results for evaluation permeability fields in \mathbf{k}' are compared individually in figure 2.12. RL policies successfully capture the optimal policy behaviour as experienced with DE results. Figure 2.13 illustrate control policies for evaluation permeability fields at ninth and twelfth indices of \mathbf{k}' . For instance, optimal policy for \mathbf{k}'_9 which refers to increasing flow through producers in the lower region of the domain is clearly observed in its RL policies.

Computational complexity is the major limitation of using RL in simulation based control problem. For all three optimisation methods (PPO, A2C and DE), we employ multiprocessing to reduce total computational time. RL algorithms use multiple CPUs to run episodes in parallel while GPUs are used for neural network back propagation computations. Depending on resource availability and parameter tuning, different number of CPUs and GPUs are used on

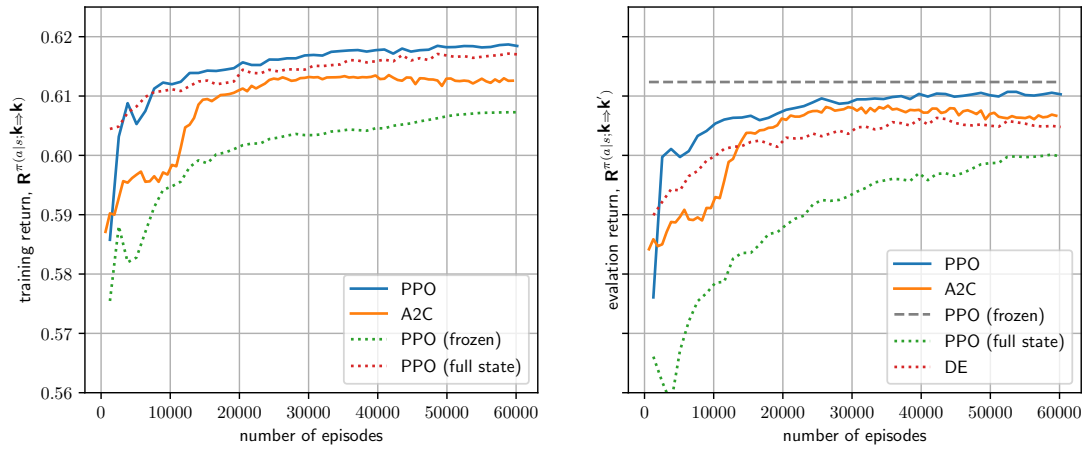


Figure 2.11: Test case 2: monitoring plots for average training return $R^{\pi(a|s; k=k)}$ (on left) and evaluation return $R^{\pi(a|s; k=k')}$ (on right) for learning process in PPO and A2C. The evaluation return value is compared with the optimisation results obtained using DE

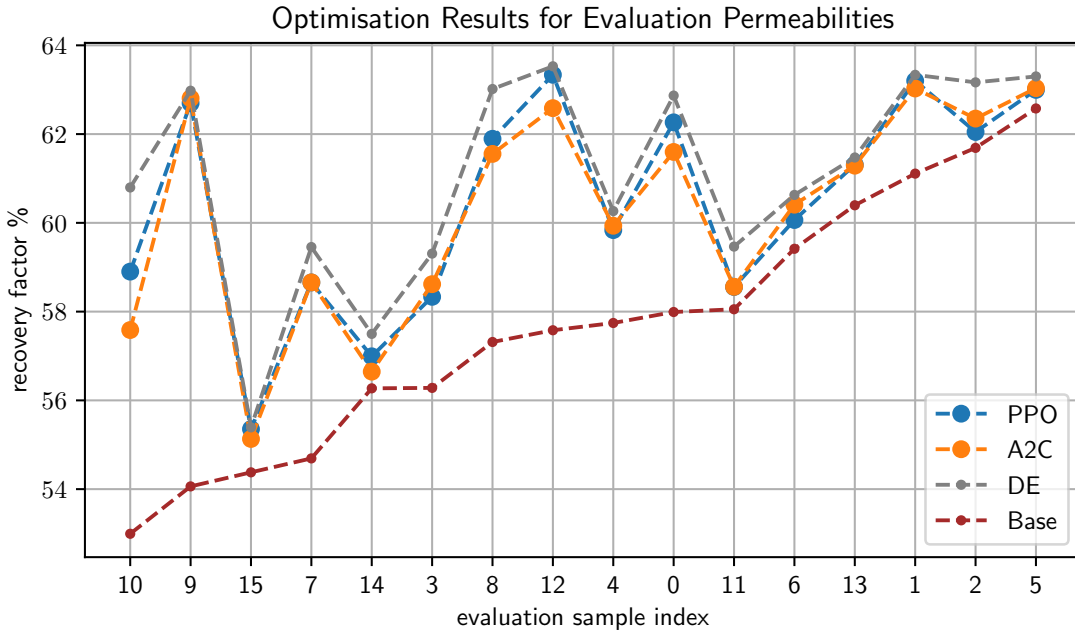
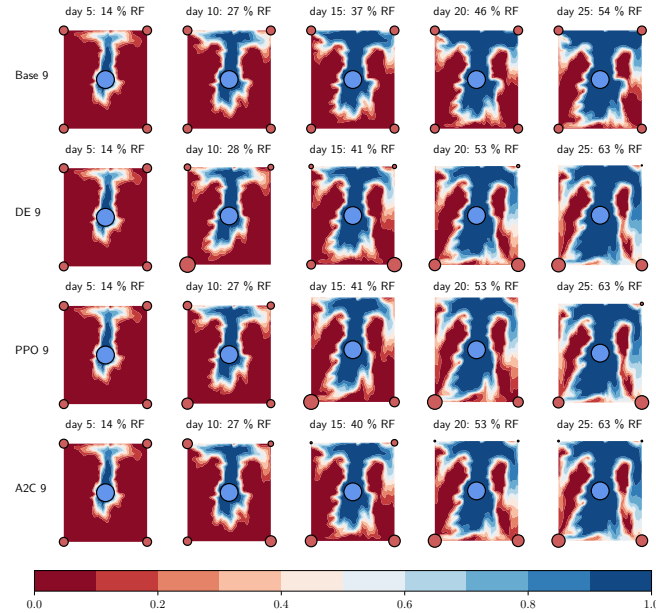
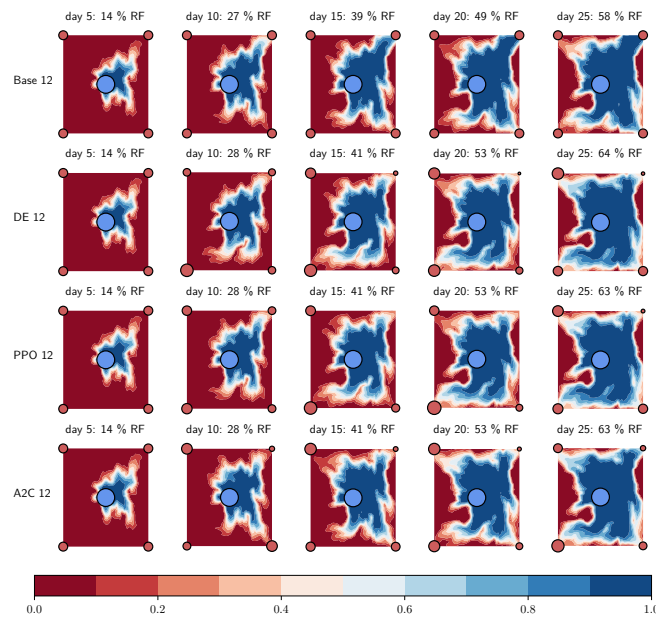


Figure 2.12: Test case 2: comparison of optimum recovery factor (in %) for each permeability value from evaluation vector \mathbf{k}' . Results of PPO, A2C and DE are compared with base control actions (all control valves equally open) results



(a) Evaluation permeability index 9



(b) Evaluation permeability index 12

Figure 2.13: Test case 2: optimal well controls for permeability values k'_9 and k'_{12} (illustrated with saturation contour plots). Producer and injector flow controls are indicated with red and blue circles, respectively. Values of flow controls are proportional to the circle radius.

Table 2.2: number of simulation runs in each algorithm

| | Test case 1 | Test case 2 |
|-----|-------------|-------------|
| PPO | 180000 | 180000 |
| A2C | 180000 | 180000 |
| DE | 2058750 | 135000 |

case basis (refer [A.3](#) for number of CPU used in each case). Furthermore, parallelism behaviour is also varied between RL and DE algorithms. In RL algorithms, simulations are run in parallel just as in the case of DE. However, in RL algorithms, neural networks are backpropagated synchronously at the end of each iteration which causes extra computational time in waiting and data distribution. In order to compare computational efforts irrespective of computational resources and parallelism behaviours, it is therefore, fair to compare number of simulation runs which is a major source of computational cost. For both cases, RL algorithms were let run for 60000 episodes which correspond to 60000 simulation runs and each such algorithm was run for three seed values (In total $3 \times 60000 = 180000$ simulation runs). For the first test case, DE algorithms consisted of 750 generations each comprising 305 population size and since DE algorithm was used for all 9 samples in \mathbf{k}' , the total number of simulation runs is 2058750 ($750 \times 305 \times 9$). Similarly, for the second test case, DE comprised of total 135000 simulation runs (750 generations \times 20 population size \times 9 samples). Parameters and computational resources used for all algorithms are delineated in [A.3](#).

2.5 Conclusions

We present a case study for using model-free RL algorithms to obtain robust optimal control policy for optimal well control problems. This policy is learned under the assumption that the system is partially observable and is governed by a system of nonlinear partial differential equations. The robustness of these policies were obtained using a domain randomisation scheme that uses only few samples from a predefined uncertainty distribution by utilizing cluster analysis. Further, the optimality of these policies were successfully benchmarked against reference solutions obtained by direct optimisation using DE algorithm. We consider the current framework as a first attempt towards application of narrow AI to the field of subsurface flow control where data is only available at the well locations.

In the current study we made the following key assumptions:

- the optimal control problem is formulated in the form of Eq. (2.1) which comprises of an objective function (Eq. (2.1a)), a governing equation (Eq. (2.1b)), initial/boundary conditions (Eq. (2.1c)) and constraints (Eq. (2.1d)),
- parameter uncertainty distribution is predefined,
- transition between the partial observations of the system approximately follows the Markov property,

- effectiveness of the optimal policy $\pi^*(a|s; k \Rightarrow k')$, is inversely related to the distance $D(k, k')$.

As a result similar techniques to those presented here could be applied to other simulation-based applications as long as these assumptions are met.

In the current study, we train RL policies with a large number of simulation runs. This can be computationally intractable for large scale models with long simulations run times. In future studies, we aim to address this issue by utilizing fast surrogate modeling techniques to accelerate the reinforcement learning process.

Chapter 3

Robust Optimal Well Control using an Adaptive Multigrid Reinforcement Learning Framework

Reinforcement learning (RL) is a promising tool for solving robust optimal well control problems where the model parameters are highly uncertain and the system is partially observable in practice. However, the RL of robust control policies often relies on performing a large number of simulations. This could easily become computationally intractable for cases with computationally intensive simulations. To address this bottleneck, an adaptive multigrid RL framework is introduced which is inspired by principles of geometric multigrid methods used in iterative numerical algorithms. RL control policies are initially learned using computationally efficient low-fidelity simulations with coarse grid discretization of the underlying partial differential equations (PDEs). Subsequently, the simulation fidelity is increased in an adaptive manner towards the highest fidelity simulation that corresponds to the finest discretization of the model domain. The proposed framework is demonstrated using a state-of-the-art, model-free policy-based RL algorithm, namely the proximal policy optimization algorithm. Results are shown for two case studies of robust optimal well control problems, which are inspired from SPE-10 model 2 benchmark case studies. Prominent gains in computational efficiency are observed using the proposed framework, saving around 60-70% of the computational cost of its single fine-grid counterpart.

3.1 Introduction

Optimal control problem involves finding controls for a dynamical system such that a certain objective function is optimized over a predefined simulation time. Recently, reinforcement learning (RL) has been demonstrated as an effective method to solve stochastic optimal control problems in fields like manufacturing [3], energy [4] and fluid dynamics [5]. RL, being virtually a stochastic optimization method, involves a large number of exploration and exploitation attempts to learn the optimal control policy. As a result, the learning process for the optimal policy comprises a large number of simulations of the controlled dynamical system, which is often computationally expensive.

Various research studies have shown the effectiveness of using multigrid methods to improve the convergence rate of reinforcement learning. [52] extend Q-Learning by casting it as a multigrid method and has shown a reduction in the number of updates required to reach a given error level in the Q-function. [53] and [54] formulate the value function learning process with a Hamilton-Jacobi-Bellman equation (HJB), which is solved using algebraic multigrid methods. However, despite the effectiveness of this strategy, the HJB formulation is only feasible when the model dynamics is well defined. As a result, these methods cannot be applied to problems where the model dynamics is an approximate representation of reality. [55] used multigrid approach to compute tabular Q values for energy conservation and comfort of HVAC in buildings, which is applicable to certain simple RL problems with finite and discrete state-action space. In this paper, the aim is to present a generalized multigrid RL approach that can be applied to both discrete and continuous state and action space where HJB formulation may not be possible. For instance, when the transition in model dynamics is not necessarily differentiable and/or when the model is stochastic.

In the context of the reinforcement learning literature, the proposed multigrid learning process can be categorized as a framework for transfer learning. In transfer learning, the agent is first trained on one or more source task(s), and the acquired knowledge is then transferred to aid in solving the desired target task [56]. In the presented study, the highest fidelity simulation corresponds to the target task, which is assumed to have the fine-grid discretization. The fine-grid discretization is presumed to guarantee a good approximation of the output quantities of interest with the accuracy required by the problem at hand. On the other hand, low-grid-fidelity simulations that compromise the accuracy of these quantities correspond to source tasks. These low-grid-fidelity simulations are generated using a degree-of-freedom parameter called the grid fidelity factor (much like the study by [57]). Transfer learning is a much broader subdomain of RL that covers knowledge transfer in the form of data samples [58], policies [59], models [60], or value functions [61]. In this study, knowledge transfer is done in the form of a policy for a model-free, on-policy algorithm called proximal policy optimization (PPO). Since the policy is designed for the state and actions corresponding to the highest-fidelity simulation, a predefined mapping function is used, which maps states and actions from low-fidelity simulations to high-fidelity simulations, and vice versa. This is done by defining restriction (mapping from high-

to low-fidelity simulation) and prolongation (mapping from low- to high-fidelity simulation) operators, which are normally found in classical geometric multigrid methods.

The effectiveness of this multigrid RL framework is demonstrated for the robust optimal well control problem, which is a subject of intensive research activities in subsurface reservoir management [24, 43, 44]. Recently, several researchers have proposed the use of reinforcement learning to solve the optimal well control problem [62–64]. For this study, the dynamical system under consideration is non-linear and, in practice, is partially observable since the data is only available at a sparse set of points (i.e., well locations). Furthermore, the subsurface model parameters are highly uncertain due to the sparsity of available field data. Optimal well control problem consists of optimizing the control variables like valve openings of wells in order to maximize sweep efficiency of injector fluid throughout the reservoir life. The reservoir permeability field is considered as an uncertain model parameter for which the uncertainty distribution is known. Two test cases – both representing a distinct model parameter uncertainty and control dynamics – are used to demonstrate the computational gains of using the multigrid idea.

In summary, a multigrid reinforcement learning framework is proposed to solve the optimal well control problem for subsurface flow with uncertain parameters. This framework is essentially inspired by the principles of geometric multigrid methods used in iterative numerical algorithms. The optimal policy learning process is initiated using a low-fidelity simulation that corresponds to a coarse grid discretization of the underlying partial differential equations (PDEs). This learned policy is then reused to initialize training against a high-fidelity simulation environment in an adaptive and incremental manner. That is, the shifting from a low fidelity to higher fidelity environments is done adaptively after the convergence of the learned policy with the low fidelity environment. Due to this adaptive learning strategy, most of the initial policy learning takes place against lower-fidelity environments, yielding a minimal computational cost in the initial stages (significant part) of the reinforcement learning process. Robustness of the policy learned using this framework is finally evaluated against uncertainties in the model dynamics.

The outline of the remainder of this paper is as follows. Section 3.2 provides the description of the problem and the proposed framework to solve the robust optimal well control problem. Section 3.3 details the model parameters for the two case studies designed for demonstration. Results of the proposed framework on these two case studies are demonstrated in Sect. 3.4. Finally, Section 3.5 concludes with a summary of the research study and an outlook on future research directions.

3.2 Methodology

Fluid flow control in subsurface reservoirs has many engineering applications, ranging from the financial aspects of efficient hydrocarbon production to the environmental problems of contaminant removal from polluted aquifers [45]. In this paper, a canonical single-phase subsurface flow control problem (also referred to as robust optimal well control problem) is studied where water

is injected in porous media to displace a contaminant. This process is commonly modeled using an advection equation for tracer flow through porous media (also called Darcy flow through porous media) over the temporal domain $\mathcal{T} = [t_0, t_M] \subset \mathbb{R}$ and spatial domain $\mathcal{X} \subset \mathbb{R}^2$. In the context of fluid displacement (e.g., groundwater decontamination), the tracer corresponds to clean water injected in the reservoir from the injector wells and the non-traced fluid corresponds to the displaced contaminated water from the reservoir through producer wells. The source and sink locations within the model domain correspond to the injector and producer wells, respectively. Tracer flow models water flooding with the fractional variable $s(x, t) \in [0, 1]$ (also known as saturation). Saturation $s(x, t)$, represents the fraction which is calculated as the ratio of injected clean water mass to the displaced contaminated water mass at location $x \in \mathcal{X}$ and time $t \in \mathcal{T}$. The flow of fluid in and out of the domain is represented by $a(x, t)$, which is treated as the source / sink terms of the governing equation. The set of well locations is denoted as $x' \in \mathcal{X}'$ (where $\mathcal{X}' \subset \mathcal{X}$). In other words, $a(x, t)$ is assigned to zero everywhere in the domain \mathcal{X} except the set of locations x' . The controls $a^+(x, t)$ (formulated as $\max(0, a(x, t))$) and $a^-(x, t)$ (formulated as $\min(0, a(x, t))$) represent the injector and producer flow controls, respectively (note that $a = a^+ + a^-$). The task of the problem under consideration is to find optimal controls $a^*(x', t)$, which is the solution of the closed-loop optimization problem defined as

$$\max_{s(\cdot), a(\cdot)} \int_{t_0}^{t_M} \left(\sum_{x'} a^-(x', t)(1 - s(x', t)) \right) dt, \quad x' \in \mathcal{X}', t \in \mathcal{T} \quad (3.1a)$$

$$\frac{ds}{dt} = \frac{1}{\phi} (a^+ + sa^- - \nabla \cdot sv), \quad x \in \mathcal{X}, t \in \mathcal{T} \quad (3.1b)$$

$$s(\cdot, t_0) = s_0, \quad v \cdot \mathbf{n} = 0, \quad (3.1c)$$

$$\sum_{x'} a^+(x', t) = - \sum_{x'} a^-(x', t) = c. \quad x' \in \mathcal{X}', t \in \mathcal{T} \quad (3.1d)$$

The objective function defined in Eq. 3.1a represents the total flow of fluid displaced from the reservoir (for example, contaminated water production) and is maximized over a finite time interval \mathcal{T} . The integrand in this function is referred to as Lagrangian term in control theory and is often denoted by $L(s, a)$. The water flow trajectory $s(x, t)$, is governed by advection Eq. 3.1b which is solved given the velocity field v , which is obtained from the Darcy law: $v = -(k/\mu)\nabla p$. The pressure $p(x, t) \in \mathbb{R}$, is obtained from the pressure equation $-\nabla \cdot (k/\mu)\nabla p = a$. Porosity $\phi(x, \cdot)$, permeability $k(x, \cdot)$, and viscosity $\mu(x, \cdot)$ are the model parameters. Permeability k , represents the model uncertainty and is treated as a random variable that follows a known probability density function \mathcal{K} with K as its domain. The initial and no-flow boundary conditions are defined in Eq. 3.1c, where \mathbf{n} denotes outward normal vector from the boundary of \mathcal{X} . The constraint defined in Eq. 3.1d represent the fluid incompressibility assumption along with the fixed total source/sink term c , which represents total water injection rate in the reservoir. In a nutshell, the optimization problem provided in

Eq. 3.1 is solved to find the optimal controls $a^*(x', t)$ such that they are robustly optimal over the entire uncertainty domain of permeability, K .

3.2.1 RL Framework

According to RL convention, the optimal control problem defined in Eq. 3.1 is modelled as a Markov decision process, which is formulated as a quadruple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$. Here, $\mathcal{S} \subset \mathbb{R}^{n_s}$ is a set of all possible states with the dimension n_s , $\mathcal{A} \subset \mathbb{R}^{n_a}$ is a set of all possible actions with the dimension n_a . The state S , is represented with the saturation $s(x, \cdot)$ and pressure $p(x, \cdot)$ values over the entire domain \mathcal{X} . The action A , is represented by an array of well control values $a(x', \cdot)$. More details of this array, like the representation of action, are presented in Sect. 3.3.3. The optimal control problem defined in Eq. 3.1 is discretized into M control steps and as a result, its solution is a set of optimal control values $a^*(x', t_1), a^*(x', t_2), \dots, a^*(x', t_M)$ where $t_0 < t_1 < t_2 < \dots < t_M$. The transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$, is assumed to follow the Markov property. That is, transition to the state $S(t_{m+1})$ is obtained by executing the actions $A(t_m)$ when in the state $S(t_m)$. Such transition function is obtained by discretizing Eq. 3.1b. For a transition from state $S(t_m)$ to state $S(t_{m+1})$, the real value reward $R(t_{m+1})$ is calculated as $R(t_{m+1}) = \mathcal{R}(S(t_m), A(t_m), S(t_{m+1}))$, where $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is the reward function. The reward function is obtained by discretizing the objective function (Eq. 3.1a) into control steps such that

$$R(t_{m+1}) = \int_{t_m}^{t_{m+1}} L(s, a) dt. \quad (3.2)$$

Optimal controls are obtained by learning a control policy function, which is defined as $\pi : \mathcal{S} \rightarrow \mathcal{A}$. This function is denoted as $\pi(A|S)$ and is generally represented by a neural network. Essentially, the control policy $\pi(A|S)$, maps a given state $S(t_m)$, to an action $A(t_m)$. For an optimal control problem, with M control steps, the goal of reinforcement learning is to find an optimal policy $\pi^*(A|S)$ such that the expected reward $G = \sum_{m=1}^M \gamma^{m-1} R(t_m)$, is maximized. Note that immediate rewards R , are exponentially decayed by the discount rate $\gamma \in [0, 1]$. The discount rate represents how myopic the learned policy is; for example, a learned policy is considered completely myopic when $\gamma = 0$. The controller, which is also referred to as an agent, follows the policy and explores various control trajectories by interacting with the environment, which consists of a transition function \mathcal{P} and a reward function \mathcal{R} . The data gathered by these control trajectories are used to update the policy towards optimality. Each such update of the policy is called a policy iteration. In RL literature, a single complete control trajectory is referred to as an episode. Essentially, RL algorithms attempt to learn the optimal policy $\pi^*(A|S)$ from a randomly initialized policy $\pi(A|S)$, by exploring the state-action space by executing a high number of episodes.

In order to represent the variability in permeability, a finite number of well spread samples is chosen from the predefined uncertainty distribution. This is achieved with a cluster analysis (see Appendix B.1 for the formulation of cluster analysis) of the distribution domain K . The sample vector $\mathbf{k} = \{k_1, k_2, \dots, k_l\}$, is constructed with samples of the distribution \mathcal{K} , which are

located nearest to the cluster centres. The policy $\pi^*(A|S)$, is learned by randomly selecting the parameter k from the training vector \mathbf{k} at the beginning of each episode. The policy return $R^{\pi(A|S)}$, is computed by averaging the returns of policy $\pi(A|S; k_i)$ (policy applied to the simulation where the permeability is set to k_i) in l simulations, which is formulated as

$$R^{\pi(A|S)} = \frac{1}{l} \sum_{i=1}^l \sum_{m=0}^{M-1} \int_{t_m}^{t_{m+1}} L(s, \pi(A|S; k_i)) dt. \quad (3.3)$$

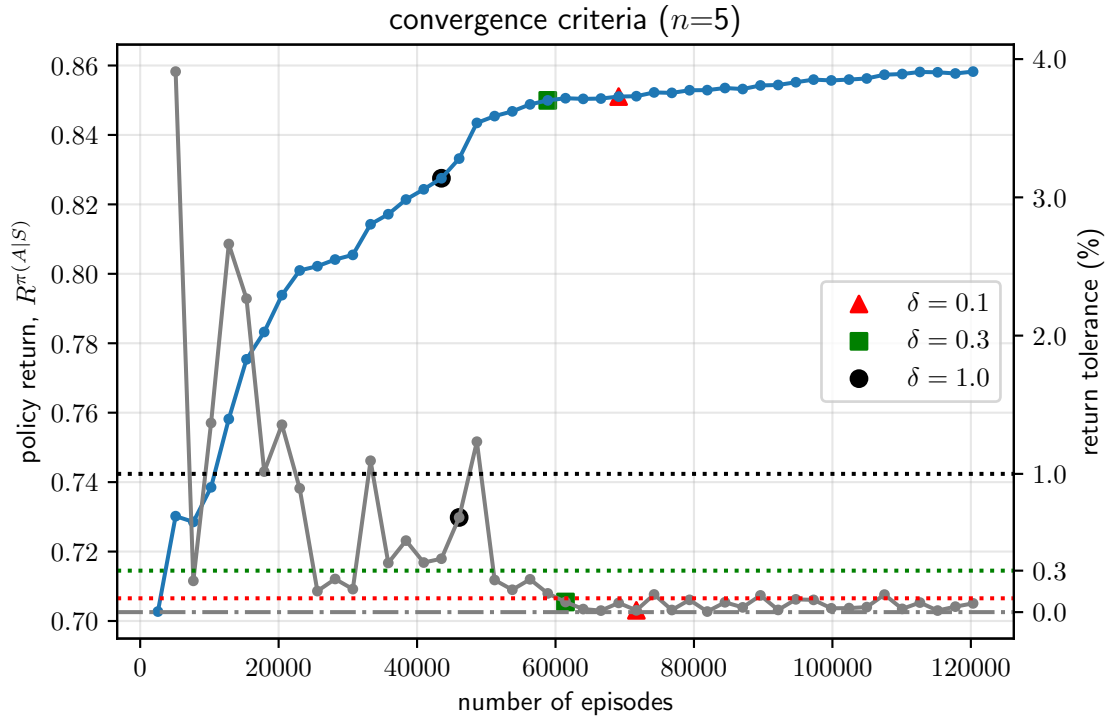
In optimal well control problems, the system is partially observable; that is, reservoir information is only available at well locations throughout the reservoir life cycle. To accommodate this fact, the agent is provided with the available observation as its state. For this study, observation is represented with a set of saturation and pressure values at the well locations x' . This is also apparent in the definition of Lagrangian term $L(s, a)$, where values s and a are taken at well locations x' , as defined in Eq. 3.1a. Note that with such a representation of states, the underlying assumption of the Markov property of the transition function is approximated. Such system is referred to as partially observable Markov decision process (POMDP). By the definition of POMDP, the policy requires the observations and actions of all previous control steps to return the action for a certain control step. However, for the presented case studies, observation from only the previous control step is observed to be sufficient for policy representation.

3.2.2 Learning Convergence Criteria

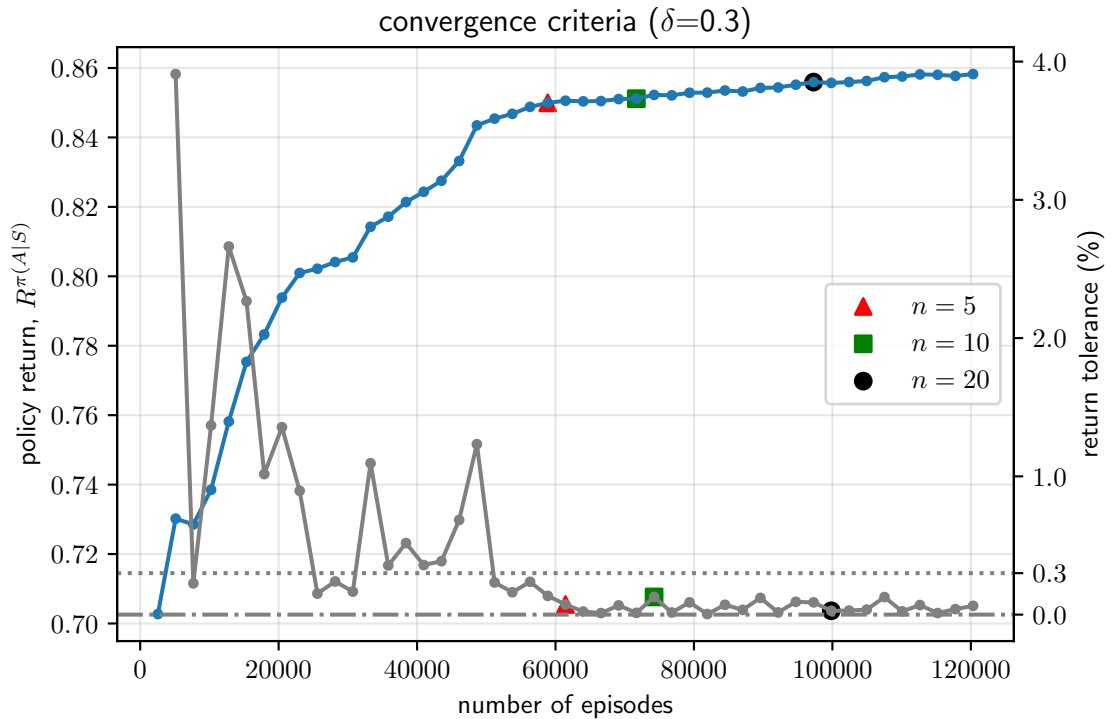
The optimal policy convergence is detected by monitoring the policy return $R^{\pi(A|S)}$, after every policy iteration. Conventionally, when this value converges to a maximum value, the optimal policy is assumed to be learned. The convergence criteria for i th policy iteration is defined as

$$\delta_i = \left| \frac{R_i^{\pi(A|S)} - R_{i-1}^{\pi(A|S)}}{\max(R_{i-1}^{\pi(A|S)}, \epsilon)} \right| < \delta, \quad (3.4)$$

where δ_i is the return tolerance at i th policy iteration, δ is the stopping tolerance and ϵ is a small non-zero number used to avoid division by zero. The convergence of policy learning process is often flat near the optimal result. For this reason, the convergence criteria defined in Eq. 3.4 is checked for the last n consecutive policy iterations. For example, if \mathbf{r} is the array of monitored values of $R^{\pi(A|S)}$ at all policy iterations, the policy $\pi(A|S)$ is considered converged when the convergence criteria are met (Eq. 3.4) for last n policy iterations is met. Algorithm 3 delineates the pseudocode for this convergence criteria. Figure 3.1 illustrates the effect of n and δ on the convergence criteria for an example of a reinforcement learning process. The policy return plot is shown in blue, where each value at the policy iteration is shown with a dot. The corresponding return tolerance is plotted in gray color, which is represented in percentage format ($\delta_i \times 100$, where δ_i is calculated from Eq. 3.4). It can be seen that the convergence criteria (denoted with markers on these plots) takes longer to satisfy when the



(a)



(b)

Figure 3.1: Plot of policy returns versus number of training episodes: **a** illustrates effect of δ on convergence criteria and **b** illustrates effect of n on convergence criteria

Algorithm 3 Learning convergence criteria

```

1: procedure ISCONVERGED( $\mathbf{r}$ ,  $n$ ,  $\delta$ )
2:   if  $length(\mathbf{r}) < n$  then return False
3:   end if
4:   compute  $\delta_i$  (Eq. 3.4) for last  $n$  values of  $\mathbf{r}$  and get its maximum  $\delta_{max}$ 
5:   if  $\delta_{max} < \delta$  then
6:     return True
7:   else
8:     return False
9:   end if
10: end procedure

```

Table 3.1: Restriction operator function for simulation parameters

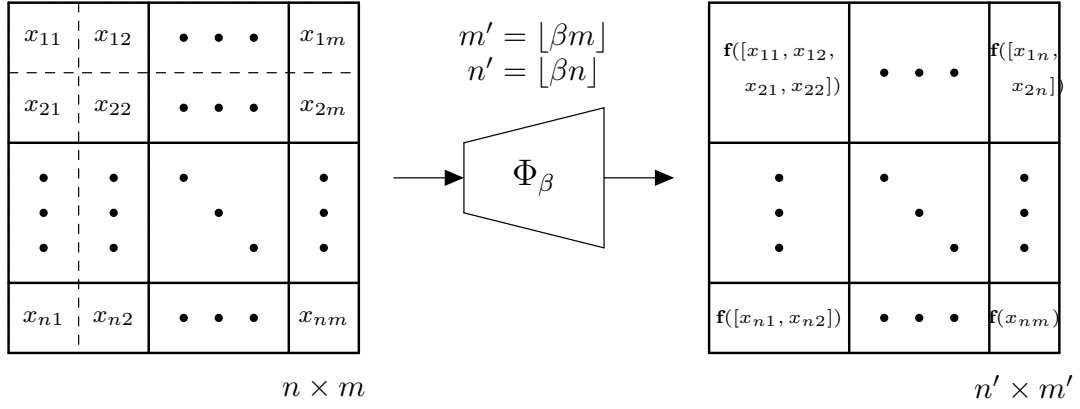
| simulation parameter | function, \mathbf{f} |
|----------------------|------------------------|
| saturation, s | mean |
| porosity, ϕ | mean |
| pressure, p | mean |
| permeability, k | harmonic mean |
| flow control, a | sum |

stopping tolerance δ , is smaller and consecutive policy iteration steps n , are higher.

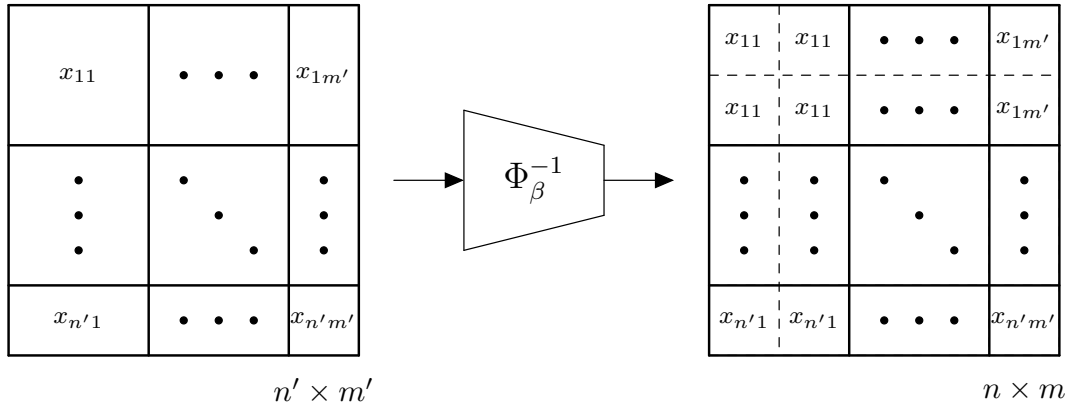
3.2.3 Adaptive Multigrid RL Framework

An adaptive multigrid RL framework is proposed where, essentially, the policies learned using lower grid fidelity environments are transferred and trained with higher-fidelity environments. The fidelity of the grid for an environment is described by the factor $\beta \in (0, 1]$. The environment with $\beta = 1$ is assumed to have the fine-grid discretization, which guarantees good approximation of fluid flow production out of the domain as defined in Eq. 3.1a. For any environment where $\beta < 1$, the size of the environment grid is coarsened by the factor of β . For example, if a high-fidelity environment where $\beta = 1$ corresponds to the simulation with grid size 64×64 , the simulation grid size is reduced to 32×32 when β is set to 0.5. Restriction operator $\Phi_\beta(\cdot)$, is used to coarsen the high fidelity simulation parameters with the factor of β . This is done by partitioning a finer grid of size $m \times n$ (corresponding to $\beta = 1$) into coarser dimensions $\lfloor \beta m \rfloor \times \lfloor \beta n \rfloor$ (corresponding to $\beta < 1$ where $\lfloor \cdot \rfloor$ is the floor operator) and computing these values of the coarse grid cell as a function \mathbf{f} , of the values in the corresponding partition. Figure 3.2(a) illustrate this restriction operator for a variable $x \in \mathbb{R}^{n \times m}$. The function \mathbf{f} , for different parameters of the reservoir simulation, are listed in table 3.1. On the other hand, the prolongation operator $\Phi_\beta^{-1}(\cdot)$, maps the coarse grid environment parameters to the fine grid as shown in Fig. 3.2(b).

A typical agent-environment interaction using this framework is illustrated in Fig. 3.3. Note that the transition function \mathcal{P} and the reward function \mathcal{R} are subscripted with β to indicate the grid fidelity of the environment. State $S(t_m)$, action $A(t_m)$ and reward $R(t_m)$ are denoted with



(a)



(b)

Figure 3.2: Illustration for the restriction operator Φ_β (a) and prolongation operator Φ_β^{-1} (b) for a parameter x

shorthand notations, S_m , A_m and R_m , respectively. Throughout the learning process, the policy is represented with states and actions corresponding to the high-fidelity grid environment. As a result, actions, and states to and from the environment, undergo the restriction Φ_β and prolongation Φ_β^{-1} operations at each time-step as shown in the environment box in the Fig. 3.3.

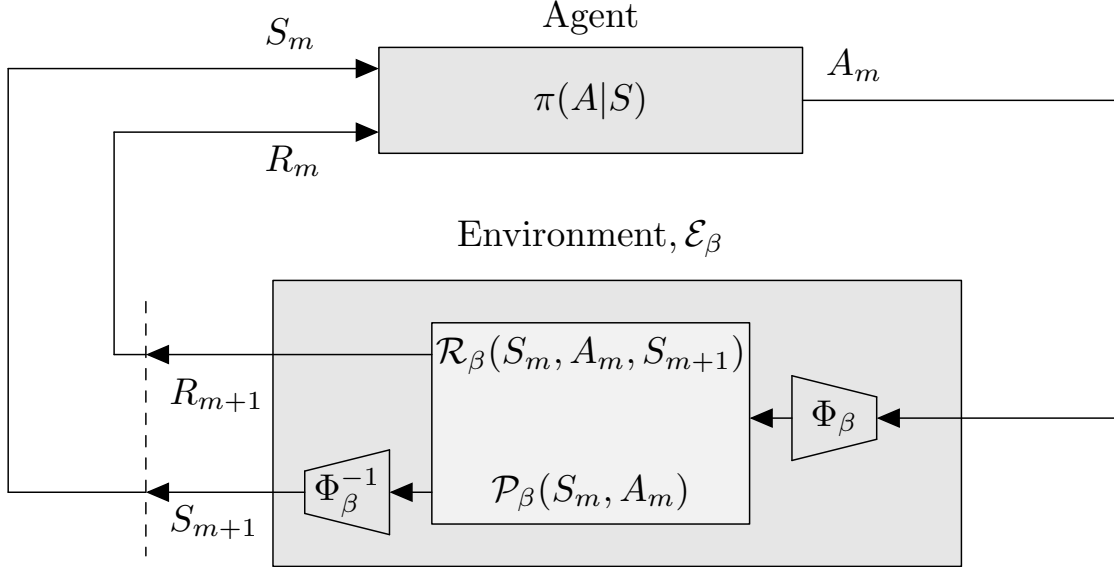


Figure 3.3: A typical agent-environment interaction in the proposed multigrid RL framework

The proposed framework is demonstrated for PPO algorithm. PPO [39] is a policy gradient algorithm that models stochastic policy $\pi_\theta(A|S)$, with a neural network (also known as the actor network). Essentially, the network parameters θ , are obtained by optimizing the objective function defined as

$$J_{ppo}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}dv(S_t, A_t), \right. \right. \\ \left. \left. \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}dv(S_t, A_t) \right) \right], \quad (3.5)$$

where $r_t(\theta) = \pi_\theta(A_t|S_t)/\pi_{\theta_{old}}(A_t|S_t)$ and θ_{old} correspond to the policy parameters before the policy update. The advantage function estimator $\hat{A}dv$, is calculated using the generalized advantage estimator [48] derived from the value function V_t . The value function estimator \hat{V}_t is learned through a separate neural network, termed as the critic network. The definitions of advantage and value functions are provided in Appendix B.2. In practice, a single neural network is used to represent both the actor and critic networks. The objective function for this integrated actor-critic network is the sum of the actor loss term (Eq. 3.5), value loss term and entropy loss term. For the purpose of maintaining brevity in our description, these latter loss terms are omitted and the policy network's objective function is treated as $J_{ppo}(\theta)$ in further discussion. However, please note that they are considered while executing the framework.

The readers are referred to [39] for a detailed definition of the policy network loss term. The algorithm 4 presents the pseudocode for the proposed multigrid RL framework. The frame-

Algorithm 4 PPO with adaptive multigrid framework

```

1: Define  $\delta$ ,  $n$  and an empty array  $\mathbf{r}$  for convergence criteria
2: Define a grid fidelity factor array  $\boldsymbol{\beta} = [\beta_1, \beta_2, \dots, \beta_m]$ , where  $\beta_m = 1$  and  $\beta_1 < \beta_2 < \dots < \beta_m$ .
3: Define an episode limit array  $\mathbf{E} = [E_1, E_2, \dots, E_m]$ , where  $E_1 < E_2 < \dots < E_m$ .
4: Define total episode count,  $e = 0$ 
5: for  $i = 1, 2, \dots, m$  do
6:   Generate the environment  $\mathcal{E}_{\beta_i}$ , with the grid fidelity factor  $\beta_i$ 
7:   for  $iteration = 1, 2, \dots$  do
8:     for  $actor = 1, 2, \dots, N$  do
9:       Run policy  $\pi_{\theta_{old}}$  in environment  $\mathcal{E}_{\beta_i}$ , for  $T$  time steps (in total,  $E$  episodes)
10:      Compute value function estimates  $\hat{V}_1, \dots, \hat{V}_T$  using critic network
11:      Compute advantage function estimates  $\hat{A}_1, \dots, \hat{A}_T$ 
12:    end for
13:    Optimize  $J_{ppo}(\theta)$  with  $K$  epochs and minibatch size  $M \leq NT$ 
14:     $\theta_{old} \leftarrow \theta$ 
15:    Compute the policy return  $R^{\pi_{\theta}(A|S)}$  and append it in  $\mathbf{r}$ 
16:     $e := e + E$ 
17:    if  $\text{IsConverged}(\mathbf{r}, n, \delta)$  or  $e \geq E_i$  then
18:      break
19:    end if
20:  end for
21: end for

```

work consists of, in total, m values of grid fidelity factor which are represented with an array $\boldsymbol{\beta} = [\beta_1, \beta_2, \dots, \beta_m]$, where $\beta_m = 1$ and $\beta_1 < \beta_2 < \dots < \beta_m$. The environment is denoted as \mathcal{E}_{β_i} , which represents the environment with the grid fidelity factor β_i . Policy $\pi_{\theta}(A|S)$ is initially learned with the environment \mathcal{E}_{β_1} , until the convergence criteria are met. The convergence criteria are checked using the algorithm 3 with predefined parameters δ and n . Upon convergence, further policy iterations are learned using the environment \mathcal{E}_{β_2} , and so on until the convergence criteria are met for the highest grid fidelity environment \mathcal{E}_{β_m} . A limit for the number of episodes to be executed at each grid level is also set. This is done by defining an episode limit array $\mathbf{E} = [E_1, E_2, \dots, E_m]$, where E_m is the total number of episodes to be executed and $E_1 < E_2 < \dots < E_m$. That is, for every environment with grid fidelity factor, β_j the maximum number of episodes to be trained is limited to E_j .

3.3 Case Studies

Two test cases are designed, representing two distinct uncertainty distributions of permeability and control dynamics. For both cases, the values for model parameters emulate those in the benchmark reservoir simulation cases, SPE-10 model 2 [51]. Table 3.2 delineates these values for test cases 1 and 2. As per the convention in geostatistics, the distribution of $\log(k)$ is

Table 3.2: Reservoir model parameters

| | case 1 | case 2 | units |
|--------------------------------|-------------|------------|----------------------|
| spatial domain \mathcal{X} | (1200×1200) | (620×1820) | ft ² |
| temporal domain \mathcal{T} | [0,125] | [0,25] | days |
| initial saturation s_0 | 0.0 | 0.0 | – |
| viscosity μ | 0.3 | 0.3 | cP |
| porosity ϕ | 0.2 | 0.2 | – |
| number of producers n_p | 31 | 14 | – |
| number of injectors n_i | 31 | 7 | – |
| total injector flow $\sum a^+$ | 2304 | 9072 | ft ² /day |

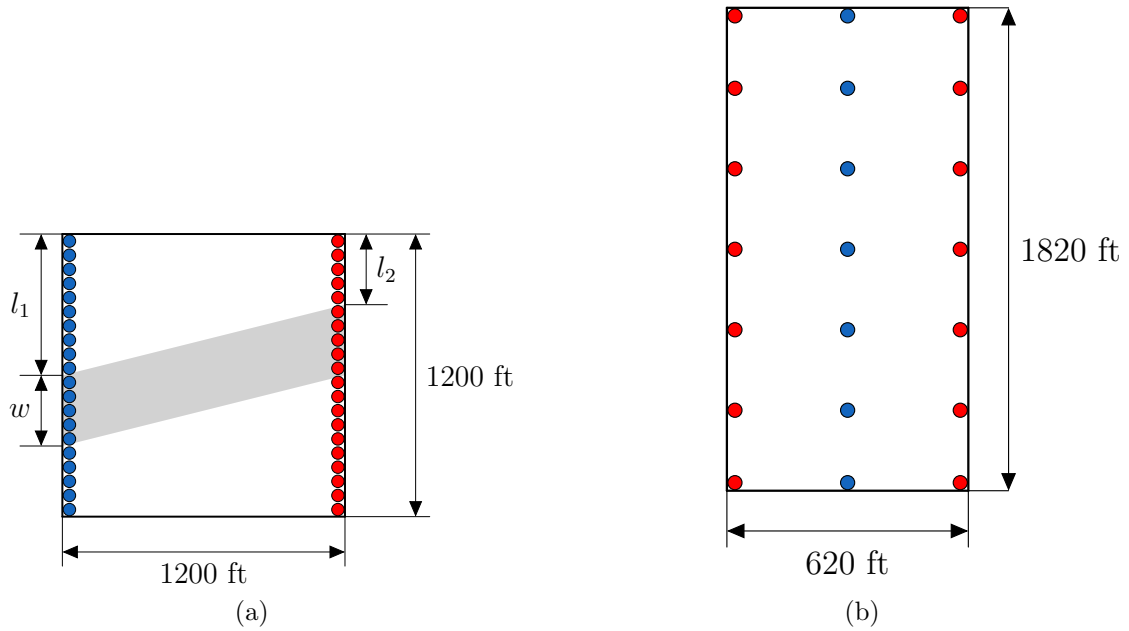


Figure 3.4: Schematic of the spatial domain for test case 1 (a) and 2 (b)

assumed to be known and is denoted by \mathcal{G} . As a result, $g = \log(k)$ is treated as a random variable in the problem description defined in Eq. 3.1. The uncertainty distributions for test cases 1 and 2 are indicated with \mathcal{G}_1 and \mathcal{G}_2 , respectively.

3.3.1 Uncertainty Distribution for Test Case 1

The log-permeability uncertainty distribution for test case 1 is inspired by the case study of [44]. Figure 3.4(a) shows schematics of the spatial domain for this case. In total, 31 injector wells (illustrated with blue circles) and 31 producer wells (illustrated with red circles) are placed on the left and right edges of the domain, respectively. As illustrated in Fig. 3.4(a), a linear high-permeability channel (shown in gray) passes from the left to right side of the domain. l_1 and l_2 represent the distance from the upper edge of the domain on the left and right sides, while the width of the channel is indicated by w . These parameters follow uniform distributions defined as $w \sim U(120, 360)$, $l_1 \sim U(0, L - w)$ and $l_2 \sim U(0, L - w)$, where L is the domain

length. In other words, the random variable g follows the probability distribution \mathcal{G}_1 that is parameterized with w , l_1 and l_2 which is described as

$$g \sim \mathcal{G}_1(w, l_1, l_2).$$

To be specific, the log-permeability g at a location (x, y) is formulated as

$$g(x, y) = \begin{cases} \log(245) & \text{if } \frac{l_2 - l_1}{L}x + l_1 \leq y \leq \frac{l_2 - l_1}{L}x + l_1 + w, \\ \log(0.14) & \text{otherwise,} \end{cases}$$

where x and y are horizontal and vertical distances from the upper left corner of the domain illustrated in Fig. 3.4(a). The values for permeability at the channel (245 mD) and the rest of the domain (0.14 mD) are inspired from Upperness log-permeability distribution peak values specified in SPE-10 model 2 case.

3.3.2 Uncertainty Distribution for Test Case 2

Test case 2 represents the uncertainty distribution of a smoother permeability field. Figure 3.4(b) illustrates reservoir domain for this case. It comprises 14 producers (illustrated with red circles) located symmetrically on the left and right edges (7 on each edge) of the domain and 7 injectors (illustrated with blue circles) located at the central vertical axis of the domain. A prior distribution F is assumed on all locations $x \in \mathcal{X}$ as

$$\begin{aligned} F(x) &= \mu + Z(x), \text{ where,} & (3.6) \\ \mathbb{E}(Z(x)) &= 0, \\ \text{Cov}(Z(x), Z(\tilde{x})) &= \sigma^2 k(x, \tilde{x}), \end{aligned}$$

where the process variance, σ , is assigned as 5 and the exponential covariance function (kernel), $k(x, \tilde{x})$, is defined as

$$k(x, \tilde{x}) = \exp \left[- \left(\frac{(x_1 - \tilde{x}_1)^2}{l_1^2} + \frac{(x_2 - \tilde{x}_2)^2}{l_2^2} \right)^{1/2} \right],$$

where the parameters l_1 and l_2 are assigned to be 620ft (width of the domain) and 62ft (10% of domain width), respectively. The posterior distribution given the observed log-permeability vector, $\mathbf{g}(x') = [g(x'_1), g(x'_2), \dots, g(x'_n)]$, where each observation corresponds to a log-permeability value of 2.41 at a well location (that is, $n = 21$ since there are, in total, 21 wells in this case). From the principle of ordinary kriging, the posterior distribution, \mathcal{G}_2 , for log-permeability at a

location $x \in \mathcal{X}$ is a normal distribution which is defined as

$$\begin{aligned} g(x) &\sim \mathcal{G}_2(\hat{g}(x), \hat{s}^2(x)), \text{ where,} \\ \hat{g}(x) &= \hat{\mu} + \mathbf{k}(x', x)^\top \mathbf{k}(x', x')^{-1} (\mathbf{g}(x') - \mathbf{1}\hat{\mu}), \\ \hat{s}^2(x) &= \sigma^2 \left[1 - \mathbf{k}(x', x)^\top \mathbf{k}(x', x')^{-1} \mathbf{k}(x', x) \right. \\ &\quad \left. + \frac{(1 - \mathbf{1}^\top \mathbf{k}(x', x')^{-1} \mathbf{k}(x', x))^2}{\mathbf{1}^\top \mathbf{k}(x', x')^{-1} \mathbf{1}} \right], \end{aligned}$$

where $\mathbf{k}(x', x)$ is the n dimensional vector whose i th value is $k(x'_i, x)$, $\mathbf{k}(x', x')$ is the $n \times n$ dimensional matrix whose value at (i, j) is $k(x'_i, x'_j)$, $\mathbf{1}$ is a n dimensional vector with all elements of one ($\mathbf{1} = [1, 1, \dots, 1]^\top$) and $\hat{\mu}$ is an estimate of the global mean μ , which is obtained from the kriging model based on the maximum likelihood estimate of the distribution $F(x)$ (from Eq. 3.6) for the observations $\mathbf{g}(x')$, and is formulated as

$$\hat{\mu} = \frac{\mathbf{1}^\top \mathbf{k}(x', x')^{-1} \mathbf{g}(x')}{\mathbf{1}^\top \mathbf{k}(x', x')^{-1} \mathbf{1}}.$$

The log-permeability distribution \mathcal{G}_2 , is created with an ordinary kriging model using the geostatistics library `gstools` [65]. In the simulation, samples of the permeability fields are obtained with a clockwise rotation angle of $\pi/8$.

3.3.3 State, Action and Reward Formulation

PPO algorithm attempts to learn the parameters θ of the policy neural network $\pi_\theta(A|S)$. The episodes (i.e., the entire simulation in the temporal domain \mathcal{T}) are divided into five control steps. Each episode timestep corresponding to a control step is denoted with t_m , where $m \in \{1, 2, \dots, 5\}$. The state S , is represented by an observation vector that consists of saturation and pressure values at well locations x' . Since the saturation values in the injector wells are always one, regardless of time t_m , they are omitted from the observation vector. Consequently, the observation vector is of the size $2n_p + n_i$ (i.e., $n_s = 93$ for test case 1 and $n_s = 35$ for test case 2). Note that this observation vector forms the input to the policy network $\pi_\theta(A|S)$. A vector of flow control values of all the injector and producer wells, denoted by A , is represented as the action. The action vector A consists of in total $n_p + n_i$ values (that is, $n_a = 62$ for test case 1 and $n_a = 21$ for test case 2). To maintain constraints defined in Eq. 3.1d, the action vector is represented by a weight vector $w \in \mathbb{R}^{n_a}$, such that $0.001 \leq w_j \leq 1$. Each weight value w_j , corresponds to the proportion of flow through the j th well. As a result, the values in the action vector are written as $(w_1, \dots, w_{n_i}, w_{n_i+1}, \dots, w_{n_i+n_p})$. Flow through j th injector A_j , is computed such that the constraint defined in Eq. 3.1d is satisfied

$$A_j = -\frac{w_j}{\sum_{i=j}^{n_i} w_j} c.$$

Table 3.3: Grid fidelity factor and corresponding grid size

| | test case 1 | test case 2 |
|----------------|----------------|----------------|
| $\beta = 1$ | 61×61 | 31×91 |
| $\beta = 0.5$ | 30×30 | 15×45 |
| $\beta = 0.25$ | 15×15 | 7×22 |

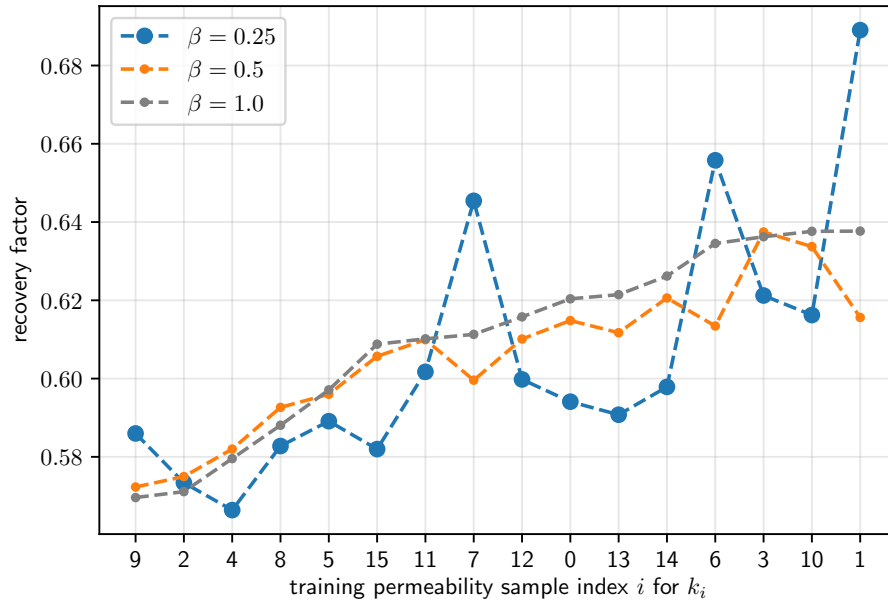
Similarly, the flow through the j th producer, A_{j+n_i} , is calculated as

$$A_{j+n_i} = \frac{w_{j+n_i}}{\sum_{j=1}^{n_p} w_{j+n_i}} c.$$

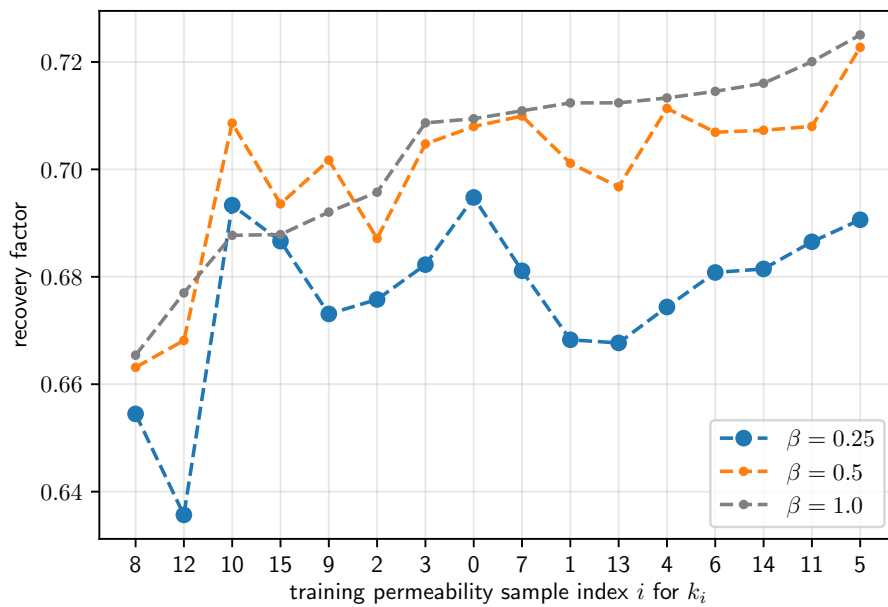
The reward function, as defined in Eq. 3.2, is divided by total pore volume ($\phi \times lx \times ly$) as a form of normalization to obtain a reward function in the range [0,1]. The normalized reward represents the recovery factor or the sweep efficiency of the contaminated fluid. Recovery factor represents the total amount of contaminants swept out of the domain. For example, the recovery factor of 0.65 means that 65% of contaminants are swept out of the domain using water flooding. To put it in the context of a groundwater decontamination problem, the optimal controls correspond to the well controls that maximize the percentage of contaminants swept out of the reservoir.

3.3.4 Multigrid Framework Formulations

The proposed framework is demonstrated using three levels of simulation grid fidelity. Note that $\beta = 1$ case corresponds to the finest level of simulation, which provides an accurate estimate of recovery factor. In this study, the aim is to exploit the simulations with $\beta < 1$, which correspond to computationally cheaper simulation run times by definition. Furthermore, $\beta < 1$ simulations are deliberately chosen such that they correspond to a recovery factor estimate that deviates from accurate $\beta = 1$ simulation estimates. Table 3.3 lists the discretization grid size corresponding to these grid fidelity factors for both test cases. Figures 3.5 (a) and 3.5 (b) plot the recovery factor estimates (with all wells open equally) for these grid fidelity factors for each permeability sample k_i , in test cases 1 and 2, respectively. The deviation from the accurate recovery factor (that is, for $\beta = 1$) for $\beta = 0.5$ and $\beta = 0.25$ can be seen for both cases. As expected, the recovery factor estimates with $\beta = 0.25$ show a higher deviation from the estimates of $\beta = 1$ compared to those of $\beta = 0.5$. To show the effectiveness of the proposed framework, the obtained results are compared with single-grid and multigrid frameworks. The results for a single-grid framework are the same as if they were obtained using the classical PPO algorithm, where the environment has a fixed fidelity factor throughout the policy-learning process. This is done by setting the grid fidelity factor array β , and episode limit array \mathbf{E} , with a single value in algorithm 4. The factor n in convergence criteria procedure (delineated in algorithm 3) is set to infinity. In other words, convergence criteria is unchecked and the policy learning takes place for a predefined number of episodes. In total, three such single-grid experiments are performed corresponding to $\beta = 0.25$, $\beta = 0.5$, and $\beta = 1.0$.



(a)



(b)

Figure 3.5: Comparison of recovery factor estimates with $\beta = 1$, $\beta = 0.5$ and $\beta = 0.25$ for test case 1 (a) and test case 2 (b)

Table 3.4: Multigrid framework experiments

| | test case 1 | test case 2 |
|--------------------------------|--------------------------------------|--|
| single grid ($\beta = 0.25$) | $\beta = [0.25]$ | $\beta = [0.25]$ |
| | $\mathbf{E} = [75000]$ | $\mathbf{E} = [150000]$ |
| | $n = \infty; \delta = 0$ | $n = \infty; \delta = 0$ |
| single grid ($\beta = 0.5$) | $\beta = [0.5]$ | $\beta = [0.5]$ |
| | $\mathbf{E} = [75000]$ | $\mathbf{E} = [150000]$ |
| | $n = \infty; \delta = 0$ | $n = \infty; \delta = 0$ |
| single grid ($\beta = 1.0$) | $\beta = [1.0]$ | $\beta = [1.0]$ |
| | $\mathbf{E} = [75000]$ | $\mathbf{E} = [150000]$ |
| | $n = \infty; \delta = 0$ | $n = \infty; \delta = 0$ |
| fixed multigrid | $\beta = [0.25, 0.5, 1.0]$ | $\beta = [0.25, 0.5, 1.0]$ |
| | $\mathbf{E} = [25000, 50000, 75000]$ | $\mathbf{E} = [50000, 100000, 150000]$ |
| | $n = \infty; \delta = 0$ | $n = \infty; \delta = 0$ |
| adaptive multigrid | $\beta = [0.25, 0.5, 1.0]$ | $\beta = [0.25, 0.5, 1.0]$ |
| | $\mathbf{E} = [25000, 50000, 75000]$ | $\mathbf{E} = [50000, 100000, 150000]$ |
| | $n = 25; \delta = 0.2$ | $n = 25; \delta = 0.2$ |

In addition, two multigrid experiments are performed to demonstrate the effectiveness of the proposed framework. The first multigrid experiment is referred to as fixed, where convergence criteria are kept unchecked just like single-grid frameworks. Multiple levels of grids are defined by setting the grid fidelity factor array β , and episode limit array \mathbf{E} , as an array of multiple values corresponding to each fidelity factor value and its corresponding episode count. In the fixed multigrid framework, policy learning takes place by updating the fidelity factor of the environment according to β without checking the convergence criteria (i.e., by setting $n = \infty$). Second, the parameters of the adaptive multigrid framework are set similarly to those used in the fixed multigrid framework, except for the convergence criteria parameters n and δ . Table 3.4 delineates the number of experiments and their corresponding parameters for test cases 1 and 2. Figure 3.6 provides visualization for the effect of fidelity factor β , on the simulation in test case 1. Figure 3.6(a) and 3.6(b) show log-permeability and saturation plots corresponding to $\beta = 0.25$, $\beta = 0.5$ and $\beta = 1.0$. Furthermore, Fig. 3.6(c) illustrates the effect of grid fidelity on simulation run time for a single episode (shown on left with a box plot of 100 simulations) and the equivalent $\beta = 1$ simulation run time for each grid fidelity factor (shown on right). Equivalent $\beta = 1$ simulation run time is defined as the ratio of average simulation run time for a grid fidelity factor β , to that corresponding to $\beta = 1$. This quantity is used as a scaling factor to convert the number of simulations for any value of β to its equivalent number of simulations as if they were performed with $\beta = 1$. Similar plots for test case 2 are shown in Fig. 3.7.

The results obtained using the proposed framework are evaluated against the benchmark optimization results. These benchmark optimal results are obtained using the differential evolution (DE) algorithm [47]. For both optimization methods (PPO and DE), multi-processing is used to reduce total computational time. However, the parallelism behavior is quite varied between the PPO and DE algorithms. For instance, neural networks are back propagated synchronously at the end of each policy iteration of PPO, which causes extra computational time in waiting

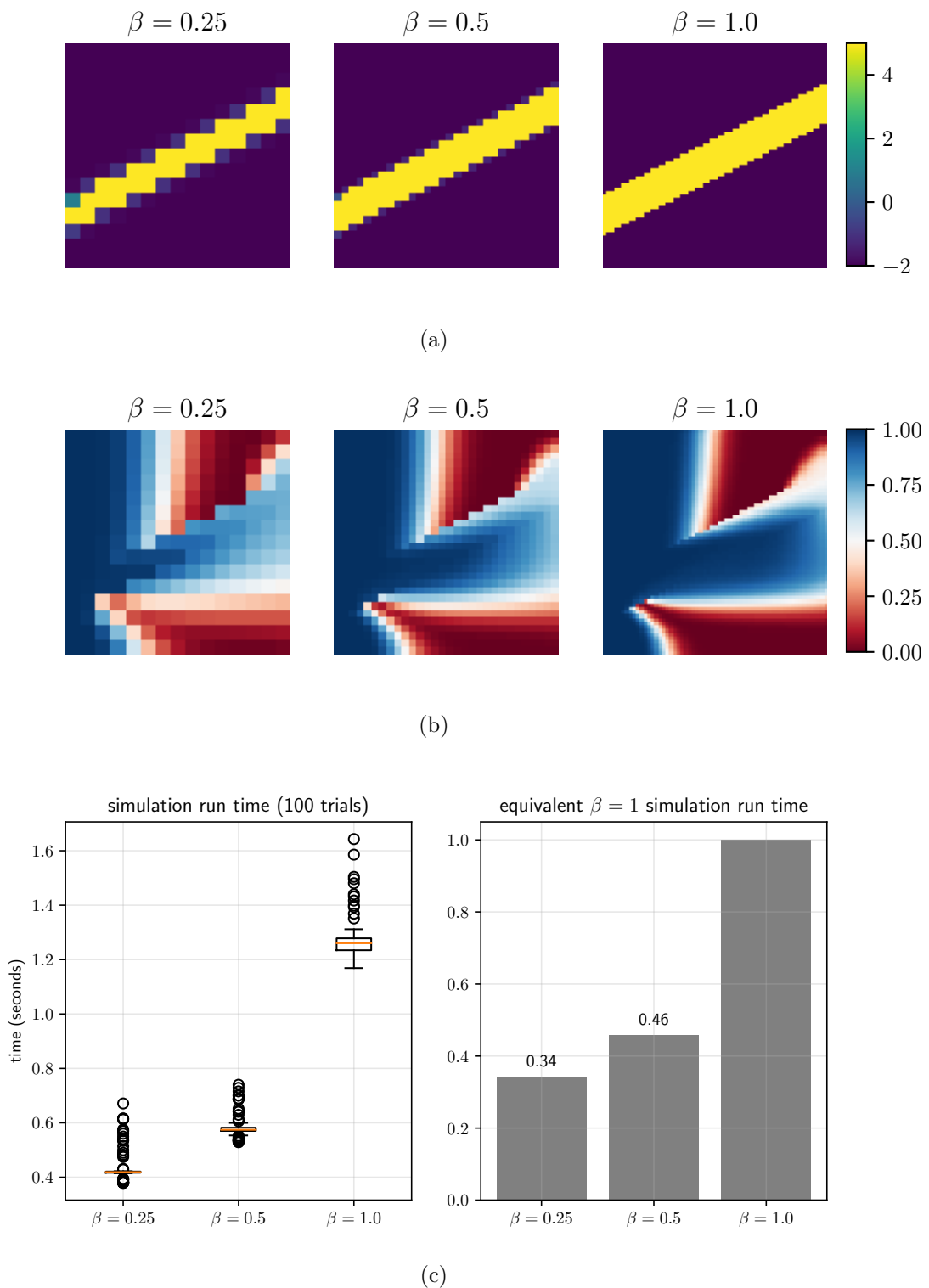


Figure 3.6: Effect of grid fidelity factor β on the environment for test case 1: **a** on a sample of log-permeability (unit: mD), **b** on corresponding saturation and **c** on simulation run time

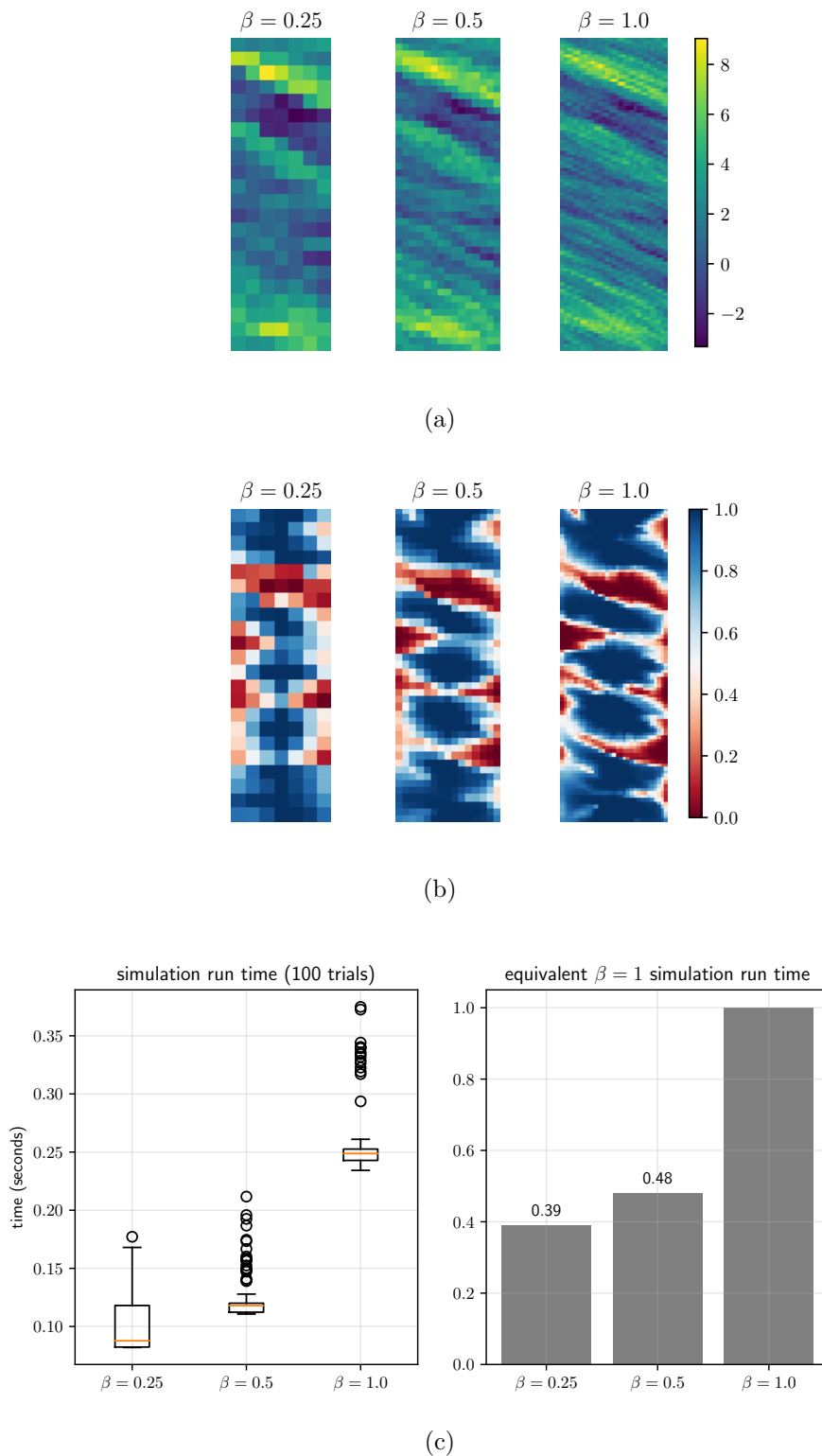


Figure 3.7: Effect of grid fidelity factor β on the environment for test case 2: **a** on a sample of log-permeability (unit: mD), **b** on corresponding saturation and **c** on simulation run time

and data distribution. For this reason, a computational cost comparison among various experiments is performed by comparing the number of simulation runs in each experiment. The PPO algorithm for the proposed framework is executed using the stable baselines library [66], while python’s SciPy [67] library is used to execute DE algorithm. Appendix B.3 delineates all the algorithm parameters used in this study.

3.4 Results

The control policy in which the injector and producer wells are equally open throughout the entire episode is called the base policy. Under such policy, the water flooding prominently takes place in the high permeability region, leaving the low permeability region swept inefficiently. The optimal policy for these test cases would be to control the producer and injector flow to mitigate this imbalance in water flooding. The optimal policy, learned using reinforcement learning for test case 1, shows on average around 12% improvement with respect to the recovery factor achieved using the base policy. While for test case 2, the average improvement is in the order of 25%.

Figure 3.8 illustrates the plots for the policy return $R^{\pi(A|S)}$, corresponding to all the frameworks listed in table 3.4 for test case 1. At the beginning of the learning process, the policy return values for single-grid framework keeps improving and eventually converge to a maximum value when the policy converges to an optimal policy. Note that for lower value of grid fidelity factor β , the optimal policy return is also low. In other words, the coarsening of simulation grid discretization also reflects in overall reduction in recovery factor. This is due to the low accuracy of the state and action representation for environments with $\beta < 1$. However, the overall computational gain is observed as a result of coarser grid sizes. The simulation run time corresponding to $\beta = 0.25$ and $\beta = 0.5$ shows a reduction of around 66% and 54% compared to $\beta = 1$. The results of multigrid frameworks are compared with the single grid framework corresponding to $\beta = 1$ which refers to the classical PPO algorithm that uses the environment with a fixed high-fidelity grid factor. As shown in the plots at the center and right of Fig. 3.8, both multigrid frameworks show convergence to the optimal policy, which is achieved using the high-fidelity single-grid framework. In the fixed multigrid framework, the fidelity factor is incremented at a fixed interval of 25,000 number of episodes. The adaptive framework is also provided with the same interval but with additional convergence check within each interval. For multigrid learning plots shown in Fig. 3.8 (center and right plots), the equivalent number of episodes corresponding to the environment with $\beta = 1$ is illustrated as a secondary horizontal axis. In this way, the computational effect of multigrid frameworks is directly compared to a single-grid framework (with $\beta = 1$). The equivalent number of $\beta = 1$ episodes corresponding to episodes with a certain value β is computed by multiplying it with the equivalent $\beta = 1$ simulation run time. For example, the number of episodes with $\beta = 0.25$ is multiplied by 0.37. For a fixed multigrid framework, it takes 45,496 equivalent $\beta = 1$ episodes to achieve an equally optimal policy that is obtained with a 75,000 number of episodes using single grid

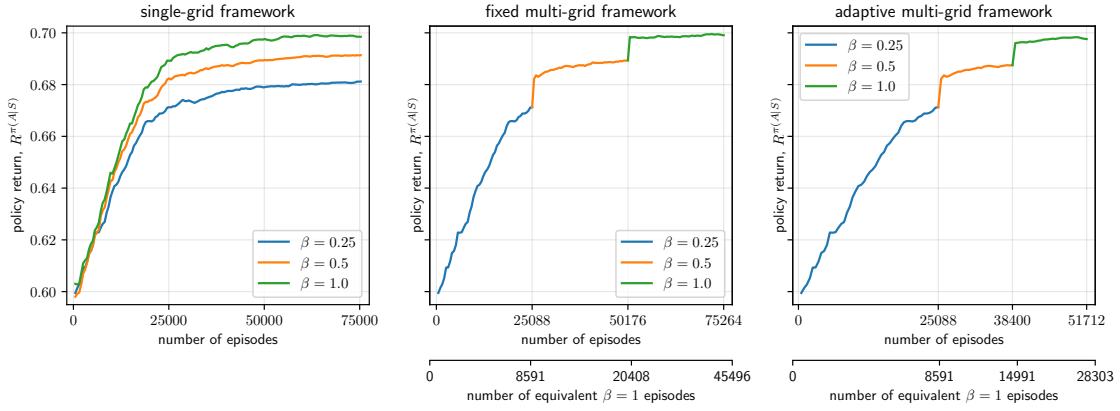
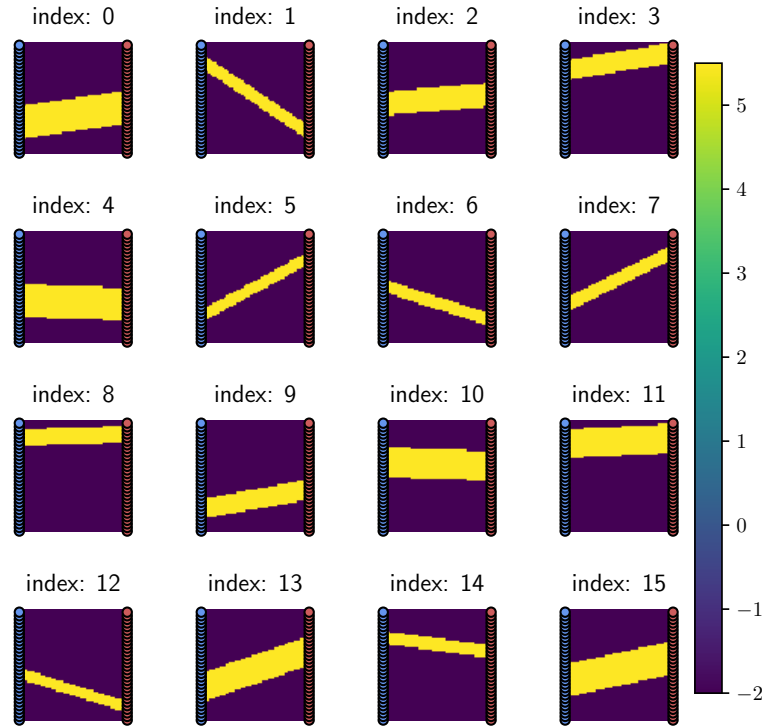


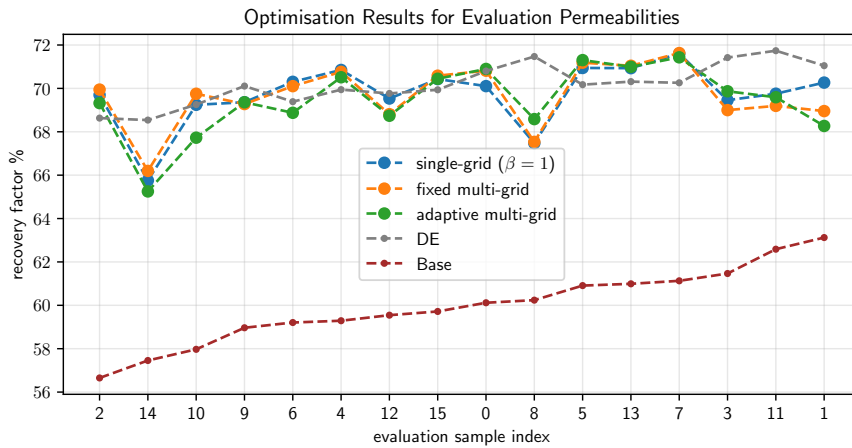
Figure 3.8: Plots of policy return versus number of episodes for test case 1

($\beta = 1$) framework. Similarly, the same is achieved with just 28,303 equivalent $\beta = 1$ episodes using the adaptive multigrid framework. In other words, around 38% and 61% reductions are observed in the simulation run time using fixed and adaptive multigrid frameworks, respectively. Further, the robustness of the policy learned using these frameworks is compared by applying it on the highest fidelity environment with random permeability samples from the distribution \mathcal{G}_1 , which were never seen during the policy learning process. Figure 3.9(a) shows the plots of these unseen permeability fields, while the corresponding results obtained using these frameworks are plotted in Fig. 3.9(b). Optimal results obtained using differential evolutionary (DE) algorithms are provided as benchmark (marked as DE in Fig. 3.9(b)). Note that DE algorithm is not a suitable method to solve the robust optimal control problem since it can provide optimal controls only for certain permeability samples as opposed to PPO algorithm where the learned policy is applicable to all samples of permeability distribution. However, DE results are used as the reference optimal results, which are achieved by direct optimization on sample-by-sample basis. Equivalence in the optimality of learned policies obtained using these three experiments can be observed from the closeness in their corresponding optimal recovery factors. Figure 3.10 demonstrates the visualization of the policy for an example of the permeability sample in case 1. In this figure, the results are shown for permeability sample index 4 from the Fig. 3.9(a) where a high permeability channel passes through the lower region of the domain. The optimal policy in this case would be to restrict flow through the injector and producer wells that are in the vicinity of the channel. The superpositioned comparison of optimal results for base case, differential evolution, single-grid framework (where $\beta = 1$), fixed multigrid framework and adaptive multigrid framework shows that the optimal policy is learned successfully using the proposed framework.

For test case 2, similar results are observed as shown in Fig. 3.11. The single-grid algorithms converge to an optimal policy in total 150,000 number of episodes. The fixed multigrid algorithm is trained with 50,000 episode interval for each grid fidelity factor as shown in the central plot in Fig. 3.11. The optimal policy is learned in 94,141 equivalent $\beta = 1$ episodes thus saving around 38% of simulation run time. The adaptive multigrid framework further reduces computational cost by achieving the optimal policy in 39,582 equivalent $\beta = 1$ episodes (simulation time



(a)



(b)

Figure 3.9: Evaluation of learned policies for test case 1: **a** evaluation samples of log-permeability distribution \mathcal{G}_1 shown with contour plots (unit: mD), **b** recovery factor (in % format) versus evaluation sample index (from **a**) plot

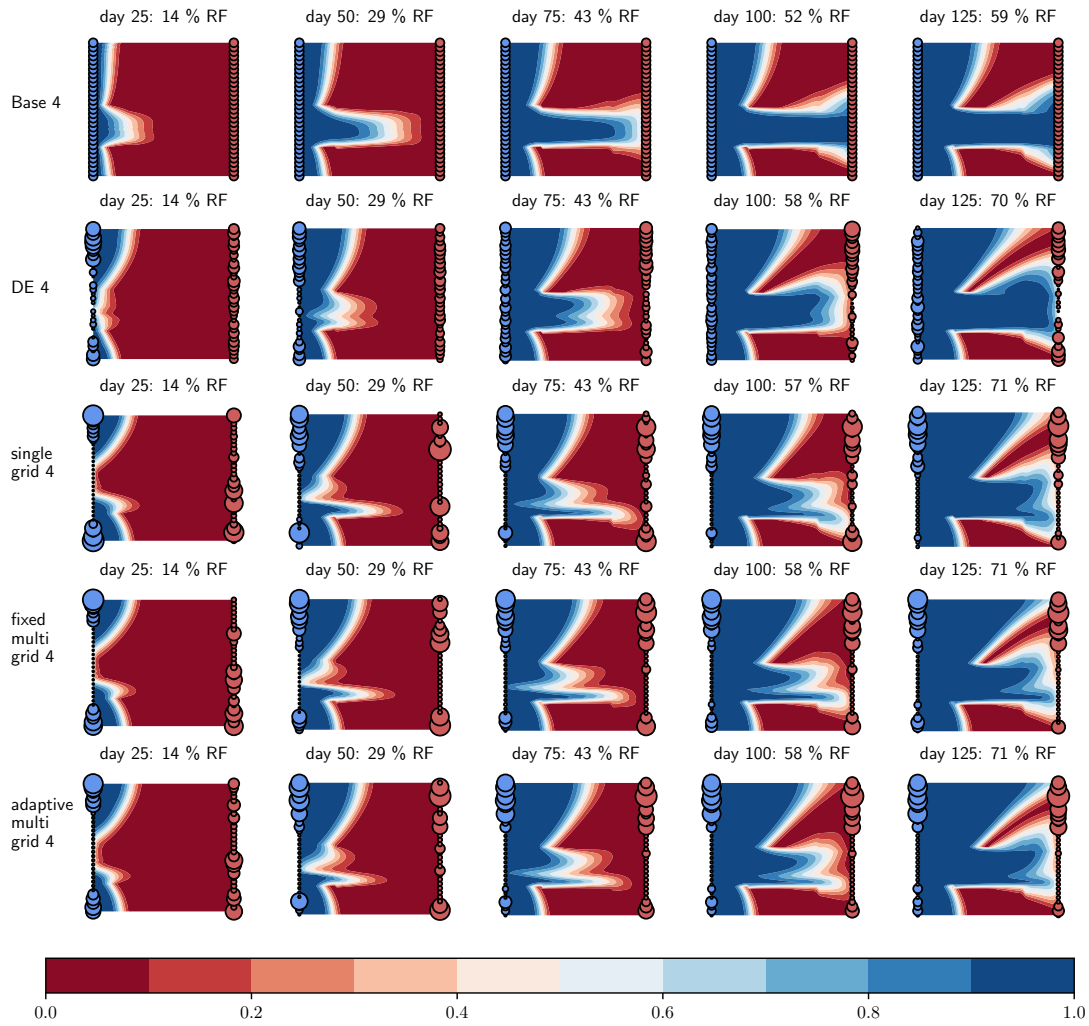


Figure 3.10: Illustration of learned optimal control policies for test case 1 using saturation contour plots

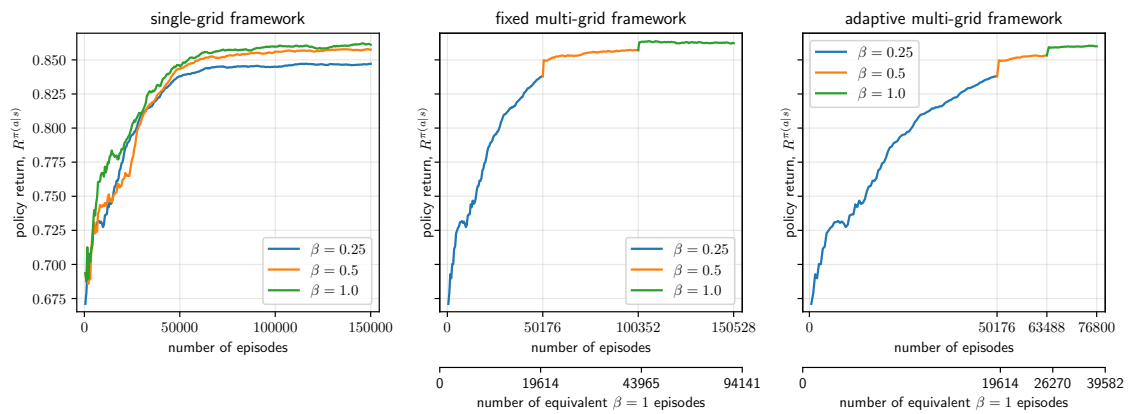
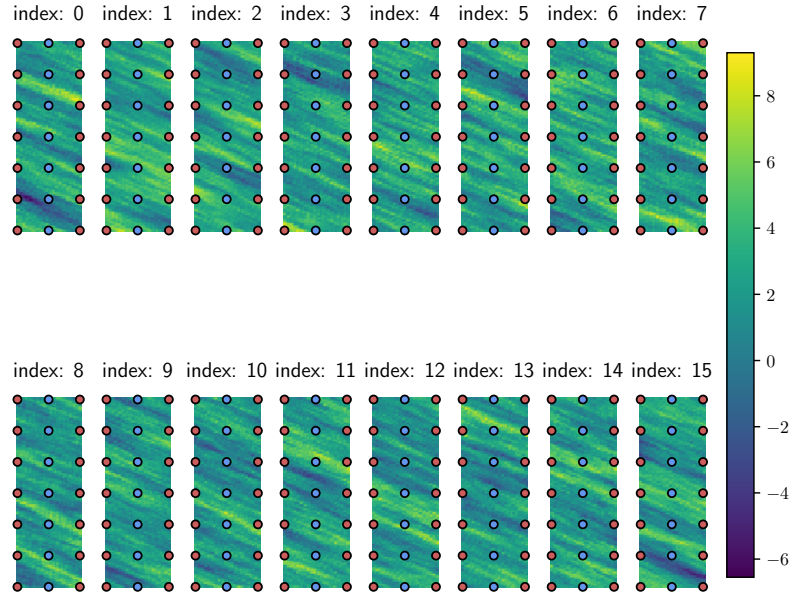


Figure 3.11: Plots of policy return versus number of episodes for test case 2

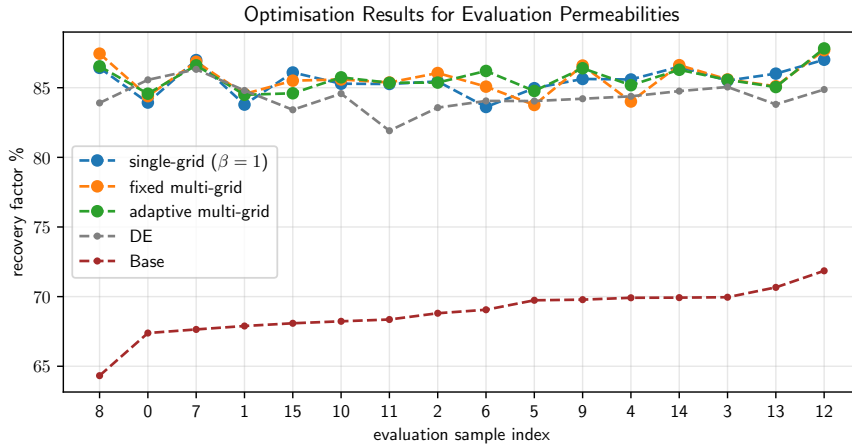
reduction of about 76% with respect to the $\beta = 1$ single-grid framework). Figure 3.12 illustrates the results of policy evaluation on unseen permeability samples from the distribution \mathcal{G}_2 . The permeability samples are shown in Fig. 3.12(a) and the optimal recovery factor corresponding to the learned policies is plotted in Fig. 3.12(b). Figure 3.13 shows the optimal controls for an example of the permeability sample index 5 from Fig. 3.12(a). The optimal policy learned using differential evolution algorithm refers to increasing the flow through injector wells which are in the low permeability region while restricting the flow through producer wells for which the water cut-off is reached. Policies learned using the RL framework take advantage of the default location and orientation of high-permeability regions. In this case, the optimal policy is achieved by controlling the well flow control such that the flow traverses through the permeability channels (that is, the flow is more or less perpendicular to the permeability orientation).

3.5 Conclusion

An adaptive multigrid RL framework is introduced to solve a robust optimal well control problem. The proposed framework is designed to be general enough to be applicable to similar optimal control problems governed by a set of time dependant nonlinear PDEs. Numerically, a significant reduction in the computational costs of policy learning is observed compared to the results of the classical PPO algorithm. In the presented case studies, 61% computational savings in simulation runtime for test case 1 and 76% for test case 2 is observed. However, note that these results are highly dependent on the right choice of the algorithm hyperparameters (e.g., δ , n , β and \mathbf{E}) which were tuned heuristically. As a future direction for this research study, the aim is to find the optimal values for β that maximize the overall computational savings. Furthermore, the policy transfer was performed sequentially in the current framework, which seemed to have worked optimally. However, to improve the generality of the proposed framework, it would be important to study the effect of the sequence of policy transfers on overall performance.



(a)



(b)

Figure 3.12: Evaluation of learned policies for test case 2: **a** evaluation samples of log-permeability distribution \mathcal{G}_2 shown with contour plots (unit: mD), **b** recovery factor (in % format) versus evaluation sample index (from **a**) plot

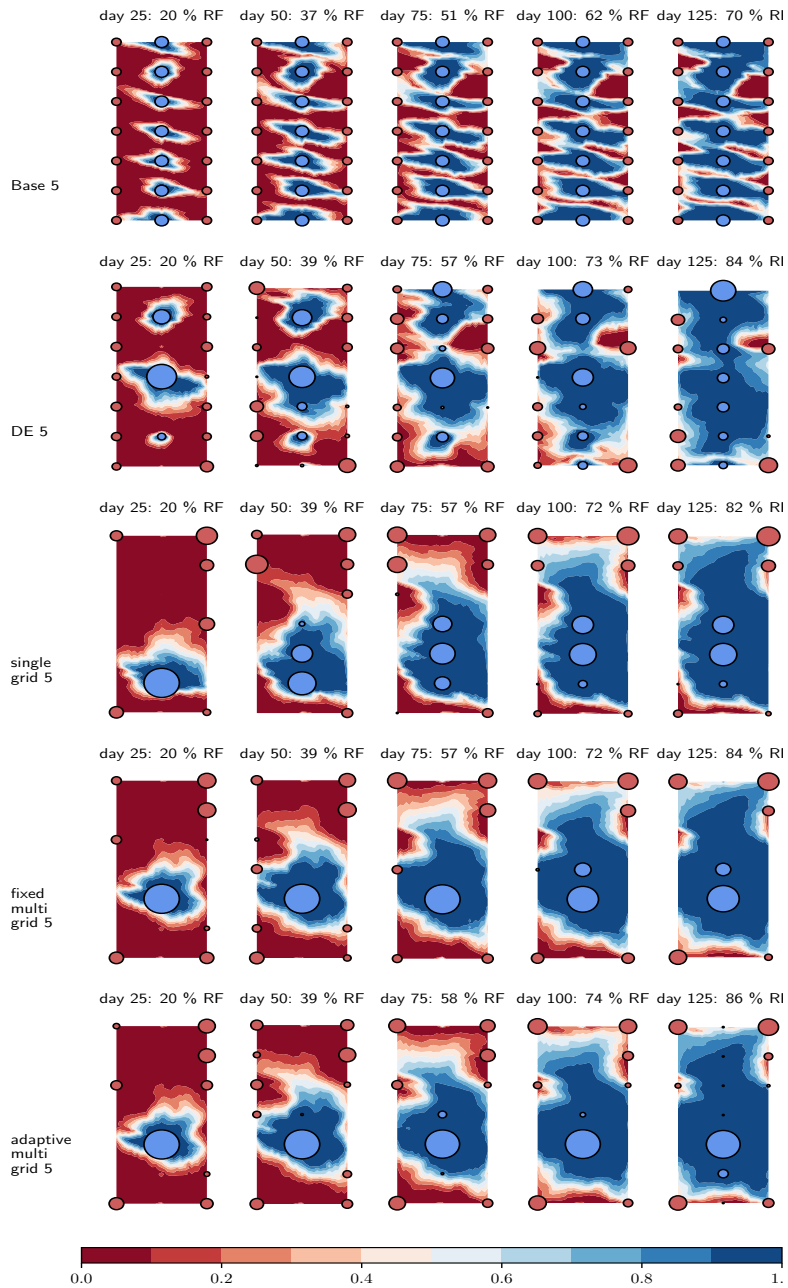


Figure 3.13: Illustration of learned optimal control policies for test case 2 using saturation contour plots

Chapter 4

A Multilevel Reinforcement Learning Framework for PDE based Control

Reinforcement learning (RL) is a promising method to solve control problems [68]. However, model-free RL algorithms are sample inefficient and require thousands if not millions of samples to learn optimal control policies. A major source of computational cost in RL corresponds to the transition function, which is dictated by the model dynamics. This is especially problematic when model dynamics is represented with coupled PDEs. In such cases, the transition function often involves solving a large-scale discretization of the said PDEs. We propose a multilevel RL framework in order to ease this cost by exploiting sublevel models that correspond to coarser scale discretization (i.e. multilevel models). This is done by formulating an approximate multilevel Monte Carlo estimate (inspired by [42]) of the objective function of the policy and/or value network instead of Monte Carlo estimates, as done in the classical framework. As a demonstration of this framework, we present a multilevel version of the proximal policy optimization (PPO) algorithm. Here, the level refers to the grid fidelity of the chosen simulation-based environment. We provide two examples of simulation-based environments that employ stochastic PDEs that are solved using finite-volume discretization. For the case studies presented, we observed substantial computational savings using multilevel PPO compared to its classical counterpart.

4.1 Introduction

Optimal control problem involves finding controls for a dynamical system (often represented by a set of partial differential equations (PDEs)) such that a certain objective function is optimized over a predefined simulation time. In recent years, we have seen a surge in research activities where reinforcement learning (RL) has been demonstrated as an effective method to solve optimal control problems in fields such as energy [4], fluid dynamics [5], and subsurface flow control [68]. The reinforcement learning process for optimal control policy often involves a large number of exploration and exploitation attempts of control trajectories. In the context of PDE-based control problems, this corresponds to a large number of simulations of the underlying model dynamics. For large-scale PDE-based problems (i.e., with high-fidelity PDE discretization), this makes RL a computationally expensive process.

Since the introduction of the multilevel Monte Carlo (MLMC) estimate as a computationally cheaper counterpart to classical Monte Carlo estimates, numerous research studies have been conducted in the application of MLMC estimates in uncertainty quantification for stochastic PDEs [69–71]. Furthermore, we also see a rise of MLMC estimate applications in certain deep learning research studies. For example, Shi and Cornish [72] present a framework for MLMC-based unbiased gradient estimation in deep latent variable models. Chada et al. [73] illustrate how the MLMC method could be applied to Bayesian inference using deep neural networks to compute expectations associated with the posterior distribution where the level corresponds to the sets of neural network parameters under consideration. In this paper, we introduce a novel multilevel framework for reinforcement learning where the learned agent interacts with environments corresponding to simulations of PDEs and the level corresponds to the grid fidelity of the PDE discretization.

We start by presenting the anatomy for classical RL algorithms, which involves estimating the Monte Carlo estimate of the objective function for policy and/or value network. Furthermore, we formulate the approximate MLMC estimation methodology used in the proposed multilevel framework. We then briefly present the mathematical framework that enables synchronized rollouts of task trajectories at different levels of the environment. The data generated through these synchronized rollouts are used to compute the approximate MLMC estimate of the objective function. Using the proposed multilevel framework, we formulate a multilevel variant of the state-of-the-art algorithm titled proximal policy optimization (PPO).

In the experiments presented, we compare the reinforcement learning process for classical and proposed multilevel PPO algorithms. The results are demonstrated for two environments for which the model dynamics is represented by stochastic partial differential equations. Furthermore, we also demonstrate the results of standard MLMC analysis to compare the MLMC and MC estimates for the PPO objective function. These environments were inspired by our research work in Dixit and ElSheikh [68]. In this study, the levels of the environment correspond to the discretization fidelity of the grid of the underlying PDEs.

The following is the outline for the rest of the paper: Section 4.2 provides the anatomy of the classical RL framework and formally defines the approximate MLMC estimation method. Section 4.3 introduces the multilevel framework for RL algorithms and further presents the multilevel PPO algorithm along with its analysis methodology. Numerical experiments to demonstrate the proposed multilevel PPO algorithm are detailed in Section 4.4, and the results of these experiments are delineated in Section 4.5. Finally, Section 4.6 concludes with a summary of the research study and an outlook on future research directions.

4.2 Background

Conventionally, the RL framework consists of the environment \mathcal{E} , which is governed by a Markov decision process described by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu \rangle$. Here, $\mathcal{S} \subset \mathbb{R}^{n_s}$ is the state-space, $\mathcal{A} \subset \mathbb{R}^{n_a}$ is the action-space, $\mathcal{P}(s'|s, a)$ is a Markov transition probability function between the current state s and the next state s' under action a and $\mathcal{R}(s, a, s')$ is the reward function. The function $\mu(s)$ returns a state from the initial state distribution if s is the terminal state of the episode (e.g., simulation terminal time); otherwise, it returns the same state s . The goal of reinforcement learning is to find the policy $\pi(a|s)$ to take an optimal action a when the state s is observed. In deep reinforcement learning, the policy is denoted $\pi_\theta(a|s)$ and is represented by a neural network with parameters θ either directly (for policy-based algorithms) or indirectly (for value-based algorithms). Learning is initiated with a random policy and then updated by exploring state-action spaces and exploiting the observed rewards in subsequent sampling steps. Each such update is referred to as a policy iteration.

Algorithm 5 Anatomy of deep reinforcement learning algorithms

- 1: **for** policy iteration = 1, 2, ... **do**
 - 2: **step 1:** Generate sequences $\{s_t, a_t, r_t\}_{t=1}^{t=T}$ using current policy $\pi_\theta(a|s)$
 - 3: **for** $t = 1, 2, \dots, T$ **do**
 - 4: generate samples s_t, a_t and r_t , where $s, a, r \sim p_\theta$
 - 5: compute Θ_t
 - 6: **end for**
 - 7: **step 2:** Compute Monte Carlo estimate of objective function $\mathbb{E}_{s,a,r \sim p_\theta} [J(s, a, r; \theta, \Theta)]$:
 - 8: $\hat{\mathbb{E}}_{s,a,r \sim p_\theta}^T [J(s, a, r; \theta, \Theta)]$
 - 9: **step 3:** Update θ using the gradient of the estimated objective function
 - 10: **end for**
-

The algorithm 5 outlines a general anatomy of deep reinforcement learning algorithms. Each policy iteration consists of three steps. First, the sequence $\{(s_1, a_1, r_1), \dots, (s_T, a_T, r_T)\}$ is generated by rolling out the current policy $\pi_\theta(a|s)$. In this stage, the RL algorithm utilizes the current policy to interact with the simulated environment by providing actions (aka. controls) and recording the observed rewards. A shorthand notation $s, a, r \sim p_\theta$, is used for the definition of random variables s, a and r . Equation 4.1 provides a detailed expansion of this shorthand

notation.

$$s, a, r \sim p_\theta \begin{cases} s \sim \mu(s) \\ a \sim \pi_\theta(a|s) \\ s' \sim \mathcal{P}(s, a) \\ r = \mathcal{R}(s, a, s') \end{cases} \quad (4.1)$$

The objective function used to calculate the gradient of the network parameters θ , is of the form $\mathbb{E}_{s,a,r \sim p_\theta}[J(s, a, r; \theta, \Theta)]$, where Θ is a set of other parameters that can vary from one algorithm to another. Appendix C.1 delineates this objective function for various algorithms. The second step consists of computing a Monte Carlo estimate of $\mathbb{E}_{s,a,r \sim p_\theta}[J(s, a, r; \theta, \Theta)]$ which is calculated using the samples generated in the first step. The notation $\widehat{\mathbb{E}}_{x \sim \mathcal{X}}^T[f(x)]$ in algorithm 5 corresponds to the Monte Carlo estimate of $\mathbb{E}_{x \sim \mathcal{X}}[f(x)]$ which is calculated as $T^{-1} \sum_{t=0}^T f(x_t)$, where x_1, \dots, x_T are random samples of the random variable $x \sim \mathcal{X}$. To maintain brevity in the description of Monte Carlo estimate, we use the same notation in the rest of the paper. Finally, in the third step, the policy is updated by updating the network parameters θ using the gradient of the estimated objective function.

4.2.1 Approximate Multilevel Monte Carlo estimation

Monte Carlo estimate of $\mathbb{E}[f(x^L)]$ for the random variable $x^L \sim \mathcal{X}^L$ is defined as

$$\mathbb{E}_{x^L \sim \mathcal{X}^L}[f(x^L)] \approx \widehat{\mathbb{E}}_{x^L \sim \mathcal{X}^L}^T[f(x^L)],$$

where T denotes the number of samples used in the estimation. Suppose that we have functions φ_L^l that approximate the random variable x^L from level L to l , $\forall l \in \{1, 2, \dots, L\}$ (note that φ_L^L is simply an identity function). Functions φ_L^l are defined so that each decrease in level l corresponds to a proportional decrease in the accuracy and cost of computing the function $f(\varphi_L^l(x^L))$. In PDE-based uncertainty quantification problems, the function f represents the quantity of interest, which implicitly contains the solution for the said PDE. The level refers to the grid discretization used during the PDE solving; that is, the grid discretization goes from coarsest to finest from level 1 to L . For such a multilevel representation of functions, MLMC estimate of $\mathbb{E}[f(x^L)]$ is defined as

$$\mathbb{E}_{x^L \sim \mathcal{X}^L}[f(x^L)] \approx \sum_{l=1}^L \widehat{\mathbb{E}}_{x^L \sim \mathcal{X}^L}^{T^l}[f(\varphi_L^l(x^L)) - f(\varphi_L^{l-1}(x^L))], \quad (4.2)$$

where T^l represents the number of samples at each level l , and the value of the function at the zeroth level is predefined at zero (that is, $f(\varphi_L^0()) \doteq 0$). The MLMC estimate is introduced by [42] as a computationally cheaper alternative to the classical Monte Carlo estimate. Readers are referred to the Appendix C.2 where we briefly explain the principle behind the computational savings in the MLMC estimation.

As described in Equation 4.2, the MLMC estimate is the telescopic sum of Monte Carlo esti-

mates of the difference term $f(\varphi_L^l(x^L)) - f(\varphi_L^{l-1}(x^L)) \forall l \in \{1, 2, \dots, L\}$, for the random variable $x^L \sim \mathcal{X}^L$. We reformulate this MLMC estimate so that we can use approximate samples at each level instead of samples from the finest level L . This is done with the following two approximations: First, we treat $\varphi_L^l(x^L)$ as a random variable $x^l \sim \mathcal{X}^l, \forall l \in \{1, 2, \dots, L\}$. Second, we replace the second difference term from $\varphi_L^{l-1}(x^L)$ to $\varphi_L^{l-1}(x^l)$. In other words, the difference term can now be computed using an approximate random variable x^l as opposed to the random variable on the finest level x^L . Furthermore, this term $\varphi_L^{l-1}(x^l)$, is denoted with \tilde{x}^{l-1} , which represents the synchronized value of x^l at the level $l-1$. We denote this synchronization process by the shorthand notation $\tilde{x}^{l-1} = \mathcal{X}^{l \Rightarrow l-1}$ as a subscript. Taking these approximations into account, we formulate the approximate MLMC estimate as follows.

$$\sum_{l=1}^L \widehat{\mathbb{E}}_{\substack{x^l \sim \mathcal{X}^l \\ \tilde{x}^{l-1} = \mathcal{X}^{l \Rightarrow l-1}}}^{T^l} [f(x^l) - f(\tilde{x}^{l-1})]. \quad (4.3)$$

Note that with this formulation we can employ the random variable x^l at each level l . This idea of using approximate samples at each level is at the heart of the proposed multilevel RL framework. In the rest of the paper, we use the Equation 4.3 notation to formulate the approximate estimate of MLMC.

4.3 Multilevel RL framework

We introduce a multilevel RL framework formulated as a tuple, $\langle \mathcal{E}, \psi_i^l, \phi_i^l \rangle$ where \mathcal{E} represents a set of multiple environments $\{\mathcal{E}^1, \mathcal{E}^2, \dots, \mathcal{E}^L\}$. An environment \mathcal{E}^L corresponds to the target task described by the tuple $\langle \mathcal{S}^L, \mathcal{A}^L, \mathcal{P}^L, \mathcal{R}^L, \mu^L \rangle$. Its corresponding sublevel tasks are represented as environments $\mathcal{E}^1, \mathcal{E}^2, \dots, \mathcal{E}^L$ such that the computational cost of \mathcal{P}^l and the accuracy of \mathcal{R}^l is lower than \mathcal{P}^{l+1} and \mathcal{R}^{l+1} for all values of $l \in \{1, \dots, L-1\}$. Furthermore, $\psi_i^l(s^l)$ is a mapping function from state on level l (denoted as s^l) to state on level l' (denoted as $s^{l'}$) and similarly $\phi_i^l(a^l)$ is a mapping function from action a^l to $a^{l'}$. The algorithm 6 outlines the anatomy of deep reinforcement learning algorithms with the proposed multilevel framework.

The first step consists of generating the sequence $\{(s_1^l, a_1^l, r_1^l), \dots, (s_{T_l}^l, a_{T_l}^l, r_{T_l}^l)\}$ on level l and its corresponding synchronized sequence $\{(\tilde{s}_1^{l-1}, \tilde{a}_1^{l-1}, \tilde{r}_1^{l-1}), \dots, (\tilde{s}_{T_l}^{l-1}, \tilde{a}_{T_l}^{l-1}, \tilde{r}_{T_l}^{l-1})\}$ on level $l-1$. The shorthand notation $s^l, a^l, r^l \sim p_\theta^l$, for generating rollouts at level l and $\tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1} = p_\theta^{l \Rightarrow l-1}$ for generating its synchronized rollouts at level $l-1$ are expanded in equation 4.4.

$$s^l, a^l, r^l \sim p_\theta^l \left\{ \begin{array}{l} s^l \sim \mu(s^l) \\ s^L = \psi_L^L(s^L) \\ a^L \sim \pi_\theta(a^L | s^L) \\ a^l = \phi_L^l(a^L) \\ s^{l'} \sim \mathcal{P}^l(s^l, a^l) \\ r^l = \mathcal{R}^l(s^l, a^l, s^{l'}) \end{array} \right. \quad \tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1} = p_\theta^{l \Rightarrow l-1} \left\{ \begin{array}{l} \tilde{s}^{l-1} = \psi_{l-1}^{l-1}(s^l) \\ \tilde{s}^L = \psi_{l-1}^L(\tilde{s}^{l-1}) \\ \tilde{a}^L \sim \pi_\theta(\tilde{a}^L | \tilde{s}^L) \\ \tilde{a}^{l-1} = \phi_{l-1}^{l-1}(\tilde{a}^L) \\ \tilde{s}^{l-1} \sim \mathcal{P}^{l-1}(\tilde{s}^{l-1}, \tilde{a}^{l-1}) \\ \tilde{r}^{l-1} = \mathcal{R}^{l-1}(\tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{s}^{l-1}) \end{array} \right. \quad (4.4)$$

Algorithm 6 Anatomy for multilevel deep reinforcement learning algorithms

```

1: for policy iteration = 1, 2, ... do
2:   step 1: Generate sequences  $\{(s_t^l, a_t^l, r_t^l), (\tilde{s}_t^{l-1}, \tilde{a}_t^{l-1}, \tilde{r}_t^{l-1})\}_{t=1}^{T_l}\}_{l=1}^{L=1}$  with policy  $\pi_\theta(a^L|s^L)$ 
3:   for level  $l = 1, 2, \dots, L$  do
4:      $s_t^l = \psi_{l-1}^l(\tilde{s}_{T_{l-1}}^{l-1})$  ▷ if  $l > 1$ 
5:     for  $t = 1, 2, \dots, T_l$  do
6:       generate samples  $s_t^l, a_t^l, r_t^l$  where  $s^l, a^l, r^l \sim p_\theta^l$ 
7:       compute  $\Theta_t^l$ 
8:       generate synchronised samples  $\tilde{s}_t^{l-1}, \tilde{a}_t^{l-1}, \tilde{r}_t^{l-1}$  ▷ if  $l > 1$ 
9:       where  $\tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1} = p_\theta^{l \Rightarrow l-1}$ 
10:      compute  $\tilde{\Theta}_t^{l-1}$  ▷ if  $l > 1$ 
11:    end for
12:  end for
13:  step 2: Compute approximate multilevel Monte Carlo estimate of objective
14:   $\mathbb{E}_{s,a,r \sim p_\theta} [J(s, a, r; \theta, \Theta)]$ :
15:  
$$\sum_{l=1}^{L=1} \widehat{\mathbb{E}}^{T_l} \int_{\substack{s^l, a^l, r^l \sim p_\theta^l \\ \tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1} = p_\theta^{l \Rightarrow l-1}}} \left[ J(s^l, a^l, r^l; \theta, \Theta^l) - J(\tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1}; \theta, \tilde{\Theta}^{l-1}) \right],$$

16:  where  $J(\tilde{s}^0, \tilde{a}^0, \tilde{r}^0; \theta, \tilde{\Theta}^0) \doteq 0$ .
17:  step 3: Update  $\theta$  using the gradient of estimated objective function
18: end for

```

Note that since the target task corresponds to the level L , the policy is now represented as $\pi_\theta(a^L|s^L)$. Consequently, during policy rollouts on a certain level l , the state s^l passes through the mapping ψ_{l-1}^l and the action obtained a^L passes through the mapping ϕ_L^l . Synchronization from level l to $l-1$ is obtained by mapping the states: $\tilde{s}^{l-1} = \psi_{l-1}^{l-1}(s^l)$. Figure 4.1 illustrates the implementations of a policy iteration in the classical and multilevel frameworks. Note that the level $l-1$ changes to l at the end of steps T_{l-1} (for $l = 2, \dots, L$) and to continue the rollouts at the level l , the state is mapped as $\psi_{l-1}^l(\tilde{s}_{T_{l-1}}^{l-1})$. The generated samples are further used to compute the approximate multilevel Monte Carlo estimate of $\mathbb{E}_{s,a,r \sim p_\theta} [J(s, a, r; \theta, \Theta)]$ which is described as

$$\sum_{l=1}^{L=1} \widehat{\mathbb{E}}^{T_l} \int_{\substack{s^l, a^l, r^l \sim p_\theta^l \\ \tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1} = p_\theta^{l \Rightarrow l-1}}} \left[J(s^l, a^l, r^l; \theta, \Theta^l) - J(\tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1}; \theta, \tilde{\Theta}^{l-1}) \right],$$

where $J(\tilde{s}^0, \tilde{a}^0, \tilde{r}^0; \theta, \tilde{\Theta}^0) \doteq 0$. Since $T_1 > T_2 > \dots > T_L$, most of the computational costs of the rollouts lean toward sub-level environments. As a result, the approximate multilevel Monte Carlo estimate requires an overall lower computational cost than the Monte Carlo estimate in the classical framework. Finally, the network parameters θ are updated using the gradient of the estimated objective function at the end of the policy iteration.

4.3.1 Multilevel PPO algorithm

We present the proposed multilevel framework for the state-of-the-art model-free algorithm, proximal policy optimization (PPO) [39]. In the context of the multilevel framework, the

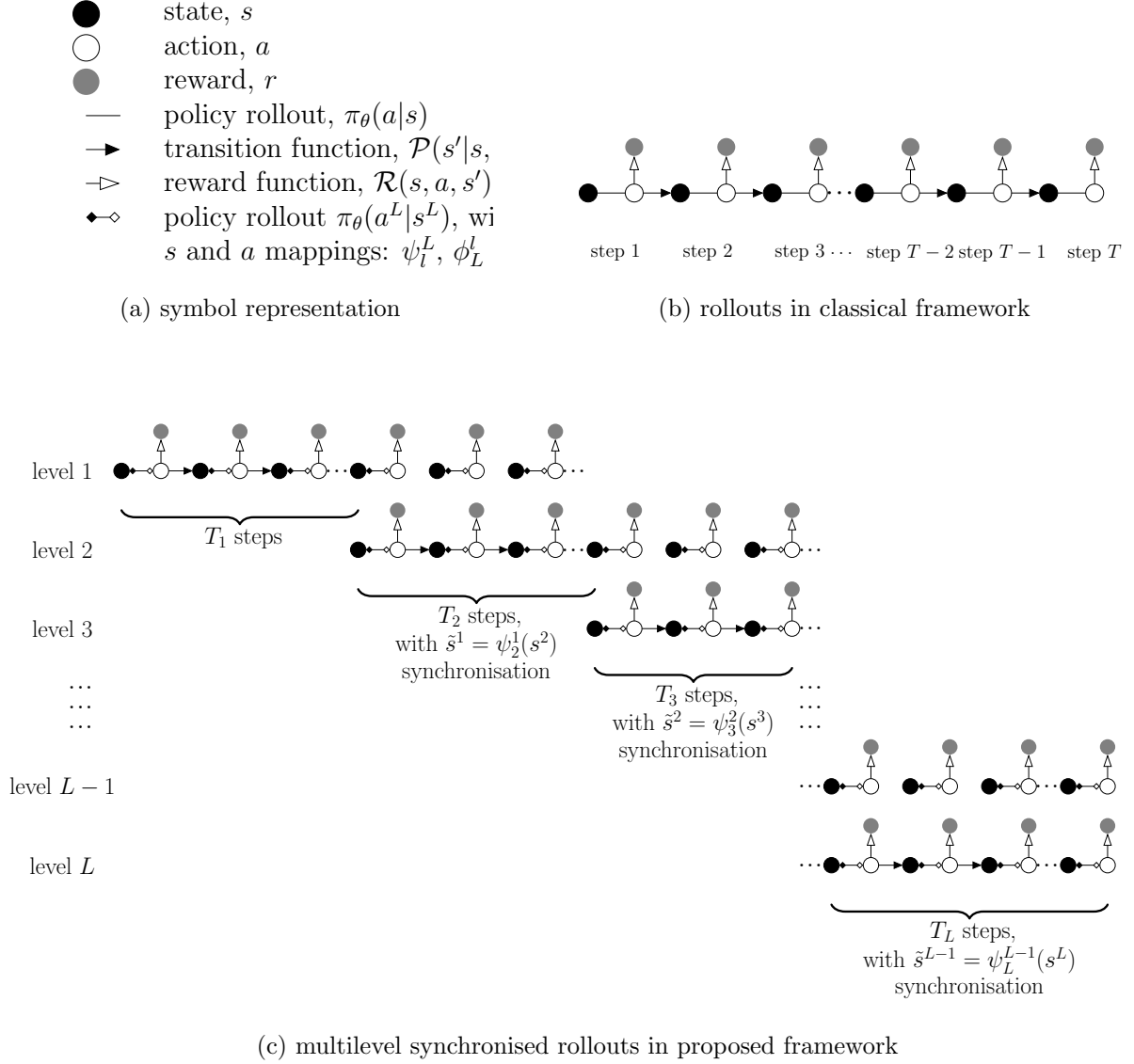


Figure 4.1: schematics of rollouts for a policy iteration

objective function is defined as

$$\begin{aligned}
 J(s, a, r; \theta, \Theta^{ppo}) = & \min (p(\theta)A(s, a), \text{clip}(\mathbf{r}(\theta), 1 - \epsilon, 1 + \epsilon)A(s, a)) \\
 & - c_v \left(r + \gamma \max_{s'} V_\theta(s') - V_\theta(s) \right)^2 \\
 & + c_e S[\pi_\theta](s).
 \end{aligned} \tag{4.5}$$

The first term of this objective function is called the surrogate policy term, where $p(\theta) = \pi_\theta(a|s)/\pi_{\theta_{old}}(a|s)$ and θ_{old} are network parameters at the beginning of the policy iteration, $A(s, a)$ is the advantage function, which is estimated using the generalized advantage estimator [48]. The second term is referred to as value function error term which correspond to learning value function $V_\theta(s)$, where γ is the discount factor. Finally, the last term $S[\pi_\theta]$, corresponds to the entropy of the learned policy, which is added to ensure sufficient exploration. Parameters Θ^{ppo} refer to the set of the following parameters: $\theta_{old}, A(s, a), \epsilon, c_v, \gamma, V_\theta, s', c_e$ and $S[\pi_\theta](s)$.

Readers are referred to the Appendix C.1 for a detailed definition of these parameters. The approximate multilevel Monte Carlo estimate of $\mathbb{E}_{s,a,r \sim p_\theta} [J(s, a, r; \theta, \Theta^{ppo})]$ is defined as

$$\sum_{l=1}^{l=L} \widehat{\mathbb{E}}^{M_l}_{s^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1} = p_\theta^{l \Rightarrow l-1}}^{s^l, a^l, r^l \sim p_\theta^l} \left[J(s^l, a^l, r^l; \theta, \Theta^{ppo^l}) - J(\tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1}; \theta, \tilde{\Theta}^{ppo^{l-1}}) \right], \quad (4.6)$$

where $J(\tilde{s}^0, \tilde{a}^0, \tilde{r}^0; \theta, \tilde{\Theta}^{ppo^0}) \doteq 0$ and M_l is the mini-batch size at level l . The algorithm 7 provides an outline for the multilevel PPO algorithm. The inputs are the same as those of

Algorithm 7 Multilevel Proximal Policy Optimization algorithm

```

1: Input:  $\mathcal{E} = \{\mathcal{E}_1, \dots, \mathcal{E}_L\}$ ,  $N$ ,  $\mathbf{T} = \{T_1, \dots, T_L\}$ ,  $\mathbf{M} = \{M_1, \dots, M_L\}$ ,  $K$ 
2: for iteration = 1, 2, ... do
3:   for actor = 1, 2, ...,  $N$  do
4:     for level  $l = 1, 2, \dots, L$  do
5:        $s^l = \psi_{l-1}^l(\tilde{s}_{T_{l-1}}^{l-1})$  ▷ if  $l > 1$ 
6:       for  $t = 1, 2, \dots, T_l$  do
7:          $s^l, a^l, r^l \sim p_\theta^l$ 
8:         compute  $\Theta^{ppo^l}$ 
9:          $\tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1} = p_\theta^{l \Rightarrow l-1}$  ▷ if  $l > 1$ 
10:        compute  $\tilde{\Theta}^{ppo^{l-1}}$  ▷ if  $l > 1$ 
11:      end for
12:    end for
13:  end for
14:  gather data  $\{ \{(s^l, a^l, r^l), (\tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1})\}_{t=1}^{t=NT_l} \}_{l=1}^{l=L}$ , from all actors
15:  optimize equation 4.6, with  $K$  epochs and minibatch size  $M_l \leq NT_l$ 
16:  update policy network parameters  $\theta$ 
17: end for

```

the classical PPO algorithm, except that multilevel variables are provided as a set of length L : environments at each level $\mathcal{E} = \{\mathcal{E}^1, \dots, \mathcal{E}^L\}$, number of actors N , number of steps at each level $\mathbf{T} = \{T^1, \dots, T^L\}$, number of batches at each level $\mathbf{M} = \{M^1, \dots, M^L\}$ (such that $NT^l \leq M^l$ and $T^1/M^1 = \dots = T^L/M^L$) and number of epochs K . Note that if the sets \mathcal{E} , \mathbf{T} and \mathbf{M} consist of a single value, this algorithm is the same as the classical PPO algorithm where the objective function is estimated using the Monte Carlo method. We implement this algorithm using a standard RL library, stable baselines 3 [66]. The implementation details are delineated in Appendix 9.

4.3.2 Multilevel PPO analysis methodology

We present an analysis methodology to compare the Monte Carlo estimate and the standard multilevel Monte Carlo estimate of the PPO objective function. The analysis methodology is adopted from [42], where the strong and weak convergences of the estimates are checked for predefined mean squared error values. For convenience of demonstration, let us consider the

following shorthand notation.

$$\begin{aligned}\widehat{\mathbb{E}}^N [Y_l] &\doteq \widehat{\mathbb{E}}^N \left[J(s^l, a^l, r^l; \theta, \Theta^{ppo^l}) - J(\tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1}; \theta, \tilde{\Theta}^{ppo^{l-1}}) \right], \\ \widehat{\mathbb{E}}^N [J_l] &\doteq \widehat{\mathbb{E}}^N_{s^l, a^l, r^l \sim p_\theta^l} \left[J(s^l, a^l, r^l; \theta, \Theta^{ppo^l}) \right].\end{aligned}$$

As a result, the multilevel Monte Carlo estimate of $\mathbb{E}[J_L]$ is described as

$$Y = \sum_{l=1}^L \widehat{\mathbb{E}}^{M_l} [Y_l]. \quad (4.7)$$

The mean squared error (MSE) for this estimator is defined as

$$\begin{aligned}MSE &= \mathbb{E}[(Y - \mathbb{E}[J_L])^2] \\ &= \mathbb{V}[Y] + (\mathbb{E}[Y] - \mathbb{E}[J_L])^2,\end{aligned}$$

where $\mathbb{V}[Y]$ is the variance of the estimator and $(\mathbb{E}[Y] - \mathbb{E}[J_L])^2$ corresponds to the bias of the estimator. A sufficient condition on $MSE \leq \varepsilon^2$, expands to $\mathbb{V}[Y] = \varepsilon^2/2$ and $(\mathbb{E}[Y] - \mathbb{E}[J_L])^2 \leq \varepsilon^2/2$. Under assumption $\mathbb{V}[Y] = \varepsilon^2/2$, the optimal number of samples at each level M_l and the corresponding total cost of the estimator C_{MLMC} are calculated as

$$M_l = 2\varepsilon^{-2} \left(\sum_{l=1}^L V_l C_l \right) \sqrt{\frac{V_l}{C_l}}, \quad (4.8)$$

$$C_{\text{MLMC}} = 2\varepsilon^{-2} \left(\sum_{l=1}^L V_l C_l \right) \sqrt{V_l C_l}, \quad (4.9)$$

where V_l corresponds to the variance estimate $\widehat{\mathbb{V}}^{N_\infty} [Y_l]$ (defined as $\widehat{\mathbb{E}}^{N_\infty} [Y_l^2] - \widehat{\mathbb{E}}^{N_\infty} [Y_l]^2$) for a large number N_∞ , of samples and C_l is the computational cost of each sample of Y_l . The weak convergence test $(\mathbb{E}[Y] - \mathbb{E}[J_L])^2 \leq \varepsilon^2/2$, is ensured by the following inequality:

$$\frac{\max_{l \in \{L-2, L-1, L\}} \widehat{\mathbb{E}}^{N_\infty} [Y_l]}{(2^\alpha - 1)} \leq \frac{\varepsilon}{\sqrt{2}}, \quad (4.10)$$

where α is assumed to be a positive coefficient that explains the decay in the values of $\widehat{\mathbb{E}}^{N_\infty} [Y_l]$ for the chosen levels in the form $\widehat{\mathbb{E}}^{N_\infty} [Y_l] = c_1 2^{-\alpha l}$. It is estimated using linear regression on $\widehat{\mathbb{E}}^{N_\infty} [Y_l]$ values. Furthermore, the multilevel estimator Y is compared with the Monte Carlo estimate corresponding to the highest level environment \mathcal{E}^L which is computed as

$$Y_{\text{MC}} = \widehat{\mathbb{E}}^M [J_L]. \quad (4.11)$$

The number of samples M and the total cost C_{MC} of the Monte Carlo estimate corresponding to the variance of the estimate $\varepsilon^2/2$ are calculated as

$$M = 2\varepsilon^{-2} \frac{V}{C}, \quad (4.12)$$

$$C_{\text{MC}} = 2\varepsilon^{-2} V \quad (4.13)$$

where V is the variance estimate $\widehat{V}^{N_\infty} [J_L]$ and C is the computational cost of each sample of J_L .

The multilevel PPO analysis is performed in parallel with learning at certain predefined intervals of policy iterations. An outline of the analysis is presented in algorithm 8. To obtain accurate

Algorithm 8 Analysis of multilevel Proximal Policy Optimization algorithm

- 1: Input: $\mathcal{E} = \{\mathcal{E}_1, \dots, \mathcal{E}_L\}$, $\varepsilon = \{\varepsilon_1, \dots, \varepsilon_n\}$, $\{C_1, \dots, C_L\}$, N_∞
 - 2: generate samples $\{s^L, a^L, r^L\}_{t=1}^{t=N_\infty}$ using the environment \mathcal{E}^L (i.e. $s^L, a^L, r^L \sim p_\theta^L$)
 - 3: generate synchronized samples $\{\{\tilde{s}^l, \tilde{a}^l, \tilde{r}^l\}_{t=1}^{t=N_\infty}\}_{l=1}^{l=L-1}$ on sublevels (where $\tilde{s}^l, \tilde{a}^l, \tilde{r}^l = p_\theta^{L \Rightarrow l}$)
 - 4: compute $\widehat{\mathbb{E}}^{N_\infty} [Y_l]$, $\widehat{\mathbb{E}}^{N_\infty} [J_l]$, $\widehat{V}^{N_\infty} [Y_l]$ and $\widehat{V}^{N_\infty} [J_l]$ using generated data
 - 5: **for** ε in ε **do**
 - 6: compute M_l (equation 4.8)
 - 7: estimate multilevel Monte Carlo estimate Y (equation 4.7)
 - 8: compute total cost C_{MLMC} (equation 4.9)
 - 9: compute M (equation 4.12)
 - 10: estimate Monte Carlo estimate Y_{MC} (equation 4.11)
 - 11: compute total cost C_{MC} (equation 4.13)
 - 12: check weak convergence (equation 4.10)
 - 13: **end for**
-

estimates of $\widehat{\mathbb{E}}^{N_\infty} [Y_l]$, $\widehat{\mathbb{E}}^{N_\infty} [J_l]$, $\widehat{V}^{N_\infty} [Y_l]$ and $\widehat{V}^{N_\infty} [J_l]$ a high number of samples N_∞ , is chosen. The samples of sequences are rolled out on the finest level L (corresponding to random variable $s^L, a^L, r^L \sim p_\theta^L$) and its synchronized samples are created in parallel on sublevels $l \in \{1, \dots, L-1\}$ (corresponding to random variables $\tilde{s}^l, \tilde{a}^l, \tilde{r}^l = p_\theta^{L \Rightarrow l}$). The notations $s^L, a^L, r^L \sim p_\theta^L$ and $\tilde{s}^l, \tilde{a}^l, \tilde{r}^l = p_\theta^{L \Rightarrow l}$ are delineated in equation 4.14 as

$$s^L, a^L, r^L \sim p_\theta^L \begin{cases} s^L \sim \mu(s^L) \\ a^L \sim \pi_\theta(a^L | s^L) \\ s'^L \sim \mathcal{P}^L(s^L, a^L) \\ r^L = \mathcal{R}^L(s^L, a^L, s'^L), \end{cases} \quad \tilde{s}^l, \tilde{a}^l, \tilde{r}^l = p_\theta^{L \Rightarrow l} \begin{cases} \tilde{s}^l = \psi_L^l(s^L) \\ \tilde{a}^L \sim \pi_\theta(a^L | s^L) \\ \tilde{a}^l = \phi_L^l(\tilde{a}^L) \\ \tilde{s}^l \sim \mathcal{P}^l(\tilde{s}^l, \tilde{a}^l) \\ \tilde{r}^l = \mathcal{R}^l(\tilde{s}^l, \tilde{a}^l, \tilde{s}^l). \end{cases} \quad (4.14)$$

The Monte Carlo and multilevel Monte Carlo estimates of the objective function of PPO are computed and compared for a set $\varepsilon = \{\varepsilon_1, \dots, \varepsilon_n\}$, of MSE accuracy values. The computational effectiveness of the multilevel estimator is demonstrated by comparing its total cost C_{MLMC} with the corresponding total cost for the Monte Carlo estimate C_{MC} for each accuracy value.

4.4 Experiments

We present two case studies of simulation environments in which the transition between states is governed by the solution of two partial differential equations that describe the incompressible flow of a single phase through porous medium. The stochasticity of the environments is attributed to an uncertain field of permeability. The governing equations for a single phase flow c of clean water, through a porous medium with porosity η , consist of the continuity equation coupled with the incompressibility condition that are defined as

$$\eta \frac{dc}{dt} = cq - \nabla \cdot cv; \quad \nabla \cdot v = q \quad \text{in } \Omega \subset \mathbb{R}^2. \quad (4.15)$$

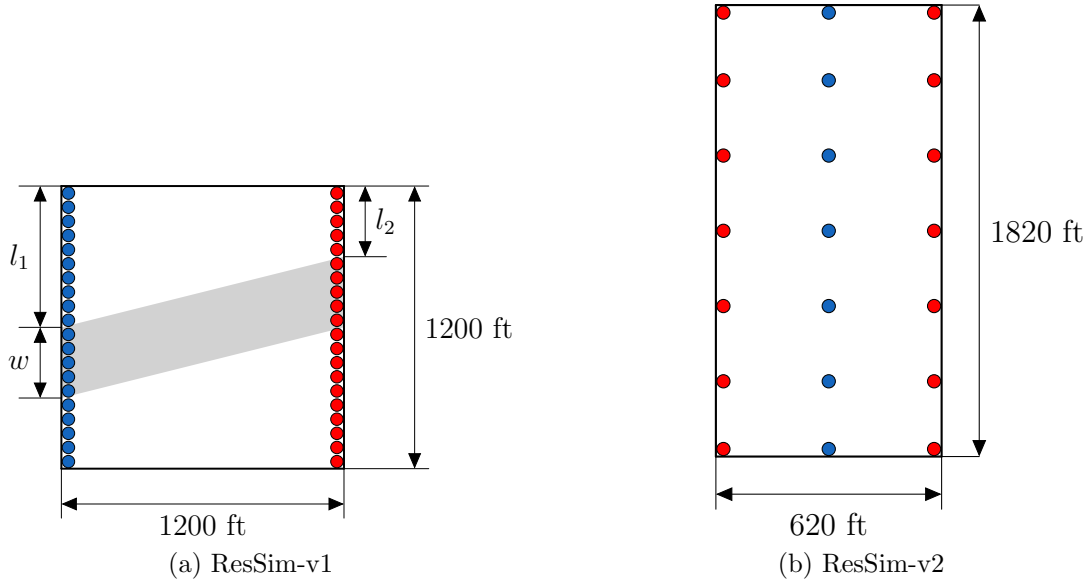
Flow velocity v and pressure p are related by Darcy's law: $v = -k/\mu \nabla p$, where k is permeability and μ is viscosity. Permeability is treated as a stochastic parameter, and its uncertainty is modeled with a predefined probability distribution. The source and sink are denoted by q , where the source corresponds to the injection rate of uncontaminated fluid (clean water) in the domain Ω , and the sink corresponds to the flow rate of the contaminated fluid at the outlet.

Two environments with distinct parameters and flow scenarios are designed for demonstration of the proposed multilevel PPO algorithm. For both cases, the parameter values emulate those of the benchmark reservoir simulations presented in SPE-10 model 2 [51]. Environments are denoted ResSim-v1 and ResSim-v2 in the rest of the paper (ResSim is a shorthand term for reservoir simulation).

4.4.1 ResSim-v1 parameters

Schematics of the domain Ω in ResSim-v1 are illustrated in Figure 4.2a. Viscosity μ is set to 0.3 cP, while porosity η is set to a constant value of 0.2. According to the convention in geostatistics, the distribution of logarithmic permeability $g = \log(k)$ is assumed to be known. This logarithmic permeability distribution for test case 1 is inspired by the case study conducted by Brouwer et al. [44]. In total, 32 injection locations (illustrated with blue circles) and 32 outlet locations (illustrated with red circles) are placed on the left and right edges of the domain, respectively. The total injection rate is set to a constant value of 2304 ft²/day. As illustrated in Figure 4.2a, a linear high-permeability channel (shown in gray) passes from the left to the right side of the domain. l_1 and l_2 represent the distance from the top edge of the domain on the left and right sides, while the width of the channel is indicated by w . These parameters follow uniform distributions defined as $w \sim U(120, 360)$, $l_1 \sim U(0, L - w)$ and $l_2 \sim U(0, L - w)$, where L is the domain length. To be specific, the logarithmic permeability g at a location (x, y) is formulated as follows:

$$g(x, y) = \begin{cases} \log(245) & \text{if } \frac{l_2 - l_1}{L}x + l_1 \leq y \leq \frac{l_2 - l_1}{L}x + l_1 + w, \\ \log(0.14) & \text{otherwise,} \end{cases}$$

Figure 4.2: schematic of the spatial domain Ω

where x and y are horizontal and vertical distances from the upper left corner of the domain, as illustrated in Figure 4.2a. The values for permeability in the channel (245 mD) and the rest of the domain (0.14 mD) are inspired from Upperness log-permeability distribution peak values specified in SPE-10 model 2 case.

4.4.2 ResSim-v2 parameters

Figure 4.2b shows the reservoir domain for ResSim-v2. It consists of 14 outlets (illustrated with red circles) located symmetrically on the left and right edges (7 on each edge) of the domain and 7 injections (illustrated with blue circles) located at the central vertical axis of the domain. The total injection rate is set at a constant value of 9072 ft²/day while viscosity and porosity are set to the same values as in ResSim-v1. The uncertainty distribution of the permeability field is considered to be smoother and spatially correlated, and is modeled as a constrained log-normal distribution. Logarithmic permeability samples are created using ordinary kriging methodology, which are constrained with a constant value of 2.41 logarithmic permeability at injection and outlet locations. The exponential variogram model used for the kriging is defined as

$$\gamma(r) = \sigma^2 \left(1 - \exp \left(-\sqrt{\left(\frac{r_x}{l_x}\right)^2 + \left(\frac{r_y}{l_y}\right)^2} \right) \right)$$

where r_x and r_y are x and y projections of the distance r . The variance of the process σ is set to 5, while the length scales l_x and l_y are set to 620 ft (width of the domain) and 62 ft (10% of domain width), respectively. The samples of permeability fields are further rotated clockwise with the angle $\pi/8$. In this study, the above-mentioned kriging process is performed using the geostatistics library gstools [65].

4.4.3 Reinforcement learning task

In the context of reinforcement learning, the state s is represented by a set of variables $\{c, k, \eta, \mu\}$, while the action a is represented by the source/sink term q at each control step. We employ finite-volume discretization of governing equations 4.15 as detailed in Aarnes, Gimse, and Lie [50] which is treated as a transition function \mathcal{P} between states at time t_m and t_{m+1} . The total time of the simulation is divided into five control steps, which form an episode with finite horizon. As a result, the task is to learn a policy $\pi_\theta(a|s)$ that selects the optimal values of q that maximize the cumulative reward defined as

$$\sum_{m=1}^5 \frac{1}{\phi|\Omega|} \int_{t_m}^{t_{m+1}} \left(\int_{\Omega} \min(q, 0)(1 - c) d\Omega \right) dt, \quad (4.16)$$

where $|\Omega|$ refers to the area of the domain. This cumulative reward refers to the sweep efficiency of the injected clean water, which ranges from 0 to 1. Furthermore, in the context of temporal difference learning, the reward at time t_m is formulated as the term inside the summation operator of equation 4.16. To represent the stochasticity of the task, a random sample of permeability is chosen from a finite set of permeabilities for each episode in the learning process. This finite set of permeability samples is achieved with a cluster analysis (please refer to Appendix C.4 for the cluster analysis formulation used in this paper). In order to demonstrate application for a partially observable system, the policy network input is replaced with an observation vector instead of the above-defined state. Here, the observation vector corresponds to values of concentration and fluid pressure at injection and outlet locations. Subsequently, the output of the policy network corresponds to the control vector, which consists of weights (with values ranging between 0.001 to 1) representing flow rates at injection and outlet locations. Note that with such representation of states, the underlying assumption of the Markov property of the transition function is approximated. Such a system is referred to as a partially observable Markov decision process (POMDP). By the definition of POMDP [74], the policy requires observations and actions from some sort of history or memory of previous control steps to return the action for a certain control step. However, for the case studies presented, observation from only the previous control step is sufficient for policy representation.

Figure 4.3 illustrates visualization of flow through the domain in ResSim-v1. The injection and outlet locations are indicated with circles in blue and red, respectively, and their radius is proportional to the flow rate. When ResSim-v1 is operated without a policy (that is, constant injection/outlet rate in all locations), most of the concentration flow takes place in the high-permeability channel, causing poor sweep efficiency in the low-permeability region. Figure 4.3a illustrates the flow scenario without a policy for a sample of permeability. The concentration flow is highlighted with blue in the domain. Consequently, the reward which refers to the sweep efficiency corresponds to the ratio of domain area highlighted in blue color to the total domain area. In other words, optimal policy refers to the choice of actions that increase the domain area in blue (i.e., swept area of the contaminate where the concentration of the clean

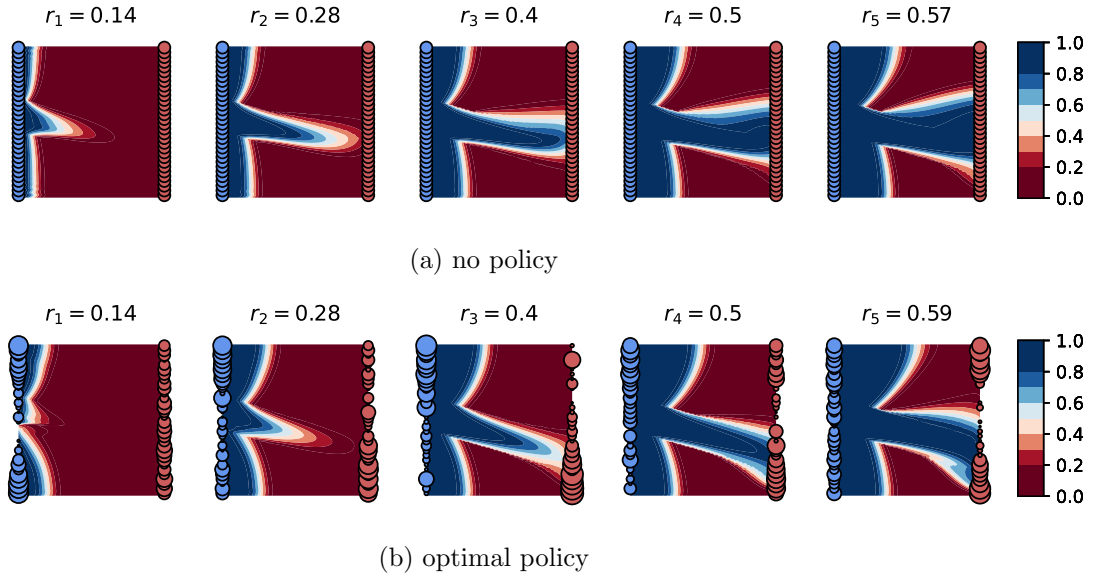


Figure 4.3: example policy visualization for ResSim-v1

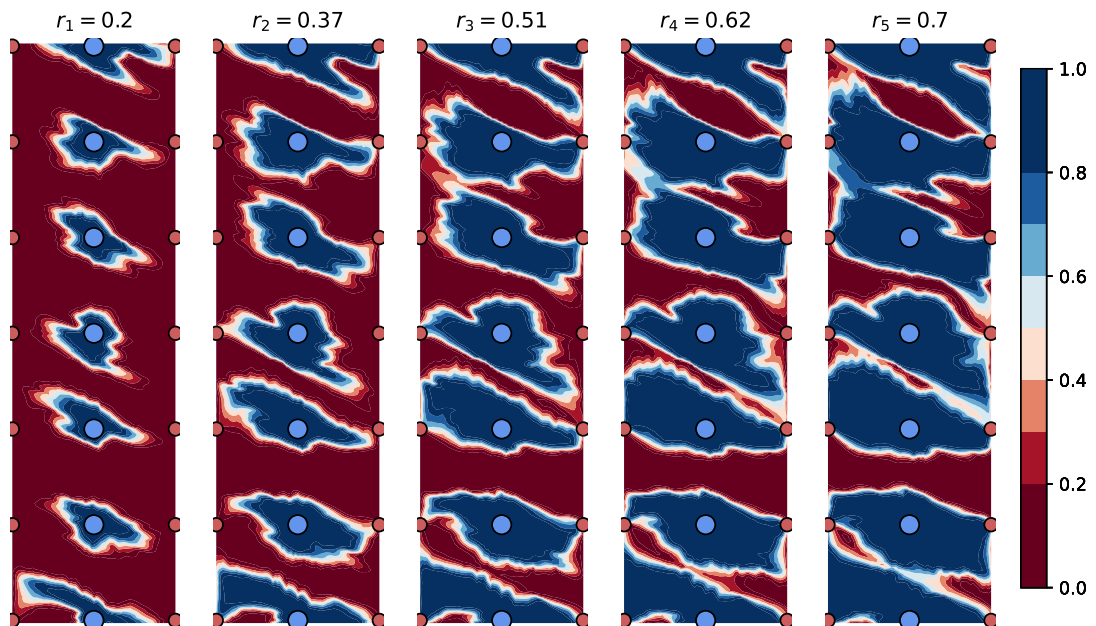
water is high). Figure 4.3b illustrates the optimal policy in which the flow through injection / outlet locations near the high permeability area near the channel is restricted. As indicated by cumulative rewards at each time, we observe an improvement in total reward at the end of the episode (from 0.57 in no policy to 0.68 with optimal policy). Similarly, Figure 4.4 provides a visualization for the ResSim-v2 environment. The optimal policy in this case is to improve the flow rate at locations near the low-permeability locations while restricting the flow rate in locations near the high-permeability region.

4.4.4 Multilevel framework formulation

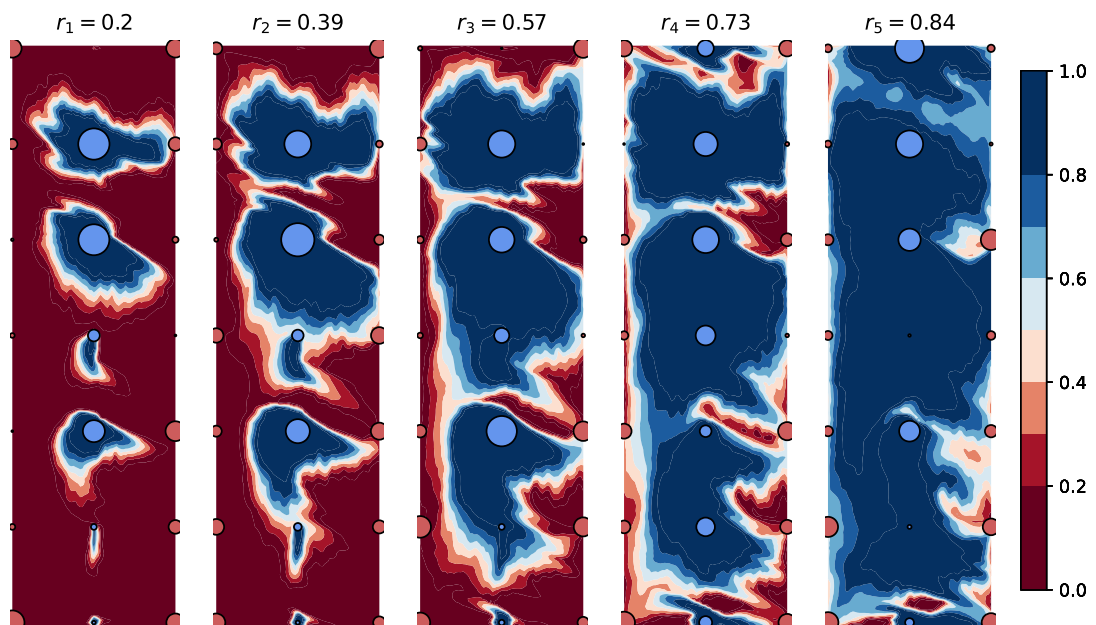
For multilevel formulation, we consider three levels of environment for the ResSim-v1 environment, where the target task is described by the environment on level 3. For ResSim-v2 environment, we consider two levels, where the target task is predefined to be at level 2. The levels for these environments correspond to the grid fidelity of the discretization scheme used to solve the governing equations. Table 4.1 delineates the grid sizes corresponding to each level in the ResSim-v1 and ResSim-v2 environments. The choice of grid fidelity corresponds to the fact

| | ResSim-v1 | ResSim-v2 |
|---------|------------------|-----------------|
| level 1 | 32×32 | 31×111 |
| level 2 | 64×64 | 73×219 |
| level 3 | 128×128 | – |

that computational cost and accuracy of model dynamics are proportional to the level. This is due to the fact that the computational cost and accuracy of a PDE are often proportional to the size of the grid. These levels of environment are chosen heuristically for demonstration purpose of the multilevel PPO algorithm. Although there could be a more systematic



(a) no policy



(b) optimal policy

Figure 4.4: example of policy visualization for ResSim-v2

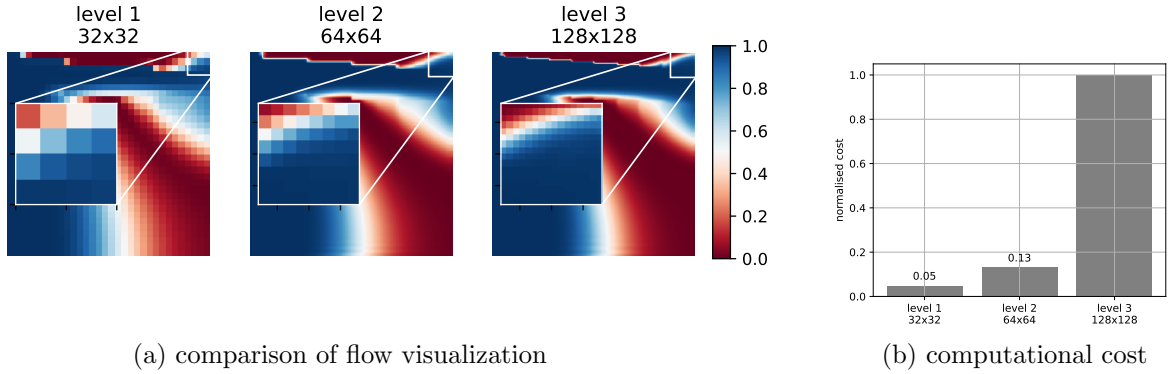


Figure 4.5: environment levels for ResSim-v1

approach to choosing these levels, we consider this to be outside the scope of this study. The state mapping function $\psi_i^{l'}$ maps the state $\{c^l, k^l, \eta^l, \mu^l\}$ for the environment on level l to the state $\{c^{l'}, k^{l'}, \eta^{l'}, \mu^{l'}\}$ for the environment on level l' . Since porosity η and viscosity μ are set to a constant throughout the domain, we do not need to map them in the function $\psi_i^{l'}$. As a result, $\psi_i^{l'}$ only maps the concentration c and the permeability k between the level l and l' . When l is larger than l' , the mapping occurs from a fine grid to a coarser grid. This is done by super-positioning a fine grid on a coarse grid and creating coarse partitions on the fine grid. The resulting values in each partition are passed through the mean function for concentration values and the harmonic mean for permeability values. On the contrary, when l' is larger than l , the mapping occurs from coarse grid to fine grid. In this case, the coarse value in each partition is simply assigned to fine grid cells in the corresponding partition. When it comes to the action mapping function $\phi_i^{l'}$, note that l' is always larger than l for the proposed multilevel framework. As a result, the action q is always mapped from a coarse grid to a finer one. The coarse to fine mapping is done with the same methodology as $\psi_i^{l'}$, except for the choice of mapping function, which is the sum of the action q . Finally, when l is the same as l' , the mapping functions $\psi_i^{l'}$ and $\phi_i^{l'}$ act as an identity function.

Figure 4.5a illustrates the comparison between flow through the domain at various levels. Comparison of the computational cost of the transition function for different levels is illustrated with a bar plot in Figure 4.5b. This computational cost is taken as an average value of 100 simulation trials to account for variability. The computational cost at each level is normalized by dividing it by that corresponding to the target task. Similar plots for the visualization of two levels of ResSim-v2 are shown in Figure 4.6.

4.5 Results

We demonstrate the effectiveness of the multilevel PPO algorithm by comparing its results with the results of the classical single-level PPO algorithm for the target task. Table 4.2 delineates the levels of environments considered in the one-level, two-level, and three-level PPO algorithm (denoted as PPO-1L, PPO-2L, and PPO-3L, respectively). Note that PPO-1L refers to the

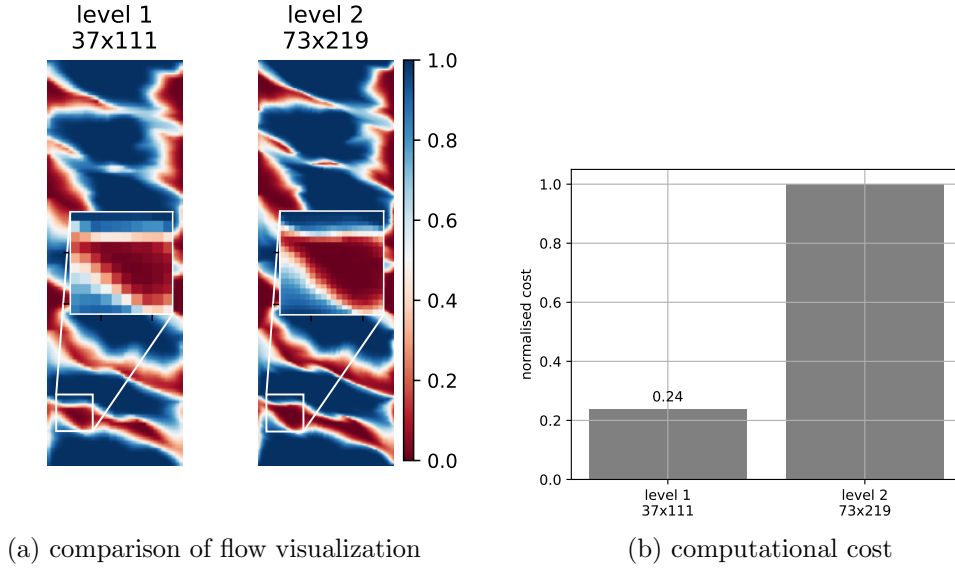


Figure 4.6: environment levels for ResSim-v2

Table 4.2: levels in each multilevel PPO experiment

| | ResSim-v1 | ResSim-v2 |
|--------|-----------|-----------|
| PPO-1L | {3} | {2} |
| PPO-2L | {2, 3} | {1, 2} |
| PPO-3L | {1, 2, 3} | — |

results of the classical single-level PPO algorithm for the target task.

4.5.1 ResSim-v1 results

First, we present the results for multilevel PPO analysis with PPO-1L, which consists of a total of 300 policy iterations. The analysis is performed every 30 iterations. Figure 4.7 illustrates the comparison of Monte Carlo and the three-level Monte Carlo estimate of the objective function with a true value that is estimated using 10^5 samples (that is, N_∞ is set to 10^5). The analysis is performed for three values of RMS accuracy: $10^{-2}, 10^{-3}, 10^{-4}$. This is done by setting $\varepsilon = \{\sqrt{10^{-2}}, \sqrt{10^{-3}}, \sqrt{10^{-4}}\}$ in the analysis. The cost terms $\{C_1, C_2, C_3\}$, which

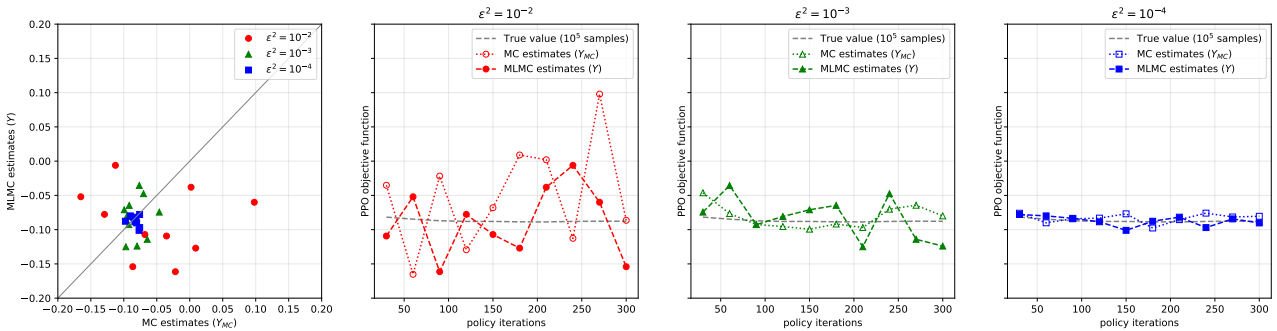


Figure 4.7: comparison of Monte Carlo and multilevel Monte Carlo estimate of PPO objective function for ResSim-v1

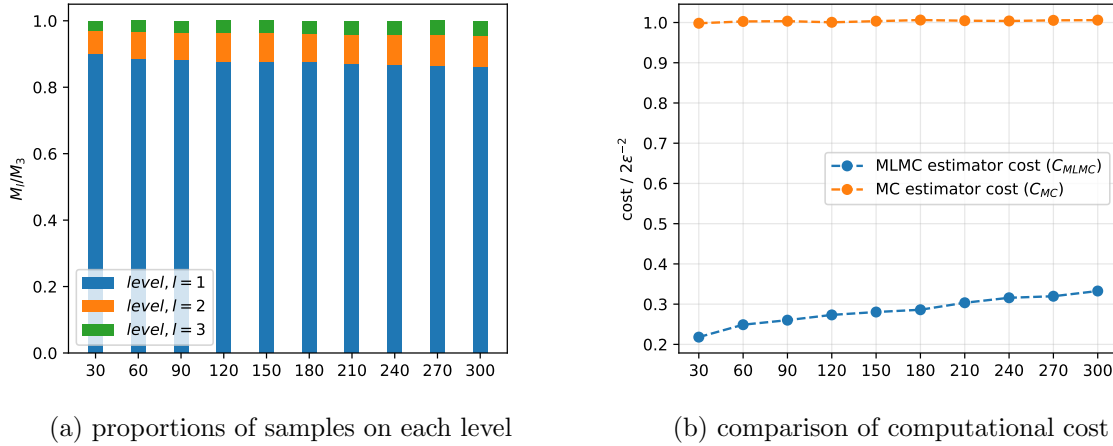


Figure 4.8: MLMC analysis results for ResSim-v1

correspond to the computational cost of each term in the multilevel Monte Carlo estimate, are set to $\{0.1, 0.33, 1.23\}$. These values are chosen from the computational cost on each level, which are illustrated in Figure 4.5b. To be specific, C_1 refers to the computational cost on level 1 (that is, 0.1). The term C_2 refers to the computational cost of the difference between synchronized samples at levels 1 and 2, as a result C_2 is set as the sum of the computational cost at levels 1 and 2 (i.e., $0.1+0.23$). Similarly, C_3 is set as the sum of the computational cost at levels 2 and 3 (that is, $0.23 + 1.0$). As can be seen in figure 4.7, we see a fairly accurate comparison between Monte Carlo and multilevel Monte Carlo estimates. Furthermore, these estimates yield more accurate values as we move towards lower values of ϵ^2 . This is because the number of samples is inversely related to ϵ^2 (as stated in equations 4.8 and 4.12). As a result, the number of samples is basically scaled up as we reduce the values of ϵ^2 . Figure 4.8a illustrates the number of optimal samples at each level of the multilevel estimator. The numbers of samples are normalized to show the proportions of the samples at each level. This is done by dividing M_l (from Equation 4.8) by M_3 for all $l \in \{1, 2, 3\}$. We see that M_2 is approximately $1/10^{th}$ of M_1 and M_3 is observed to be about half of M_2 throughout the learning process. The comparison between the computational cost of Monte Carlo and the multilevel Monte Carlo estimate is plotted in Figure 4.8 b. Here, the computational cost terms C_{MC} and C_{MLMC} are divided by $2\epsilon^{-2}$ to obtain the constant cost terms irrespective of RMS accuracy. We observe that the computational cost of the multilevel estimate takes only about 20 to 30% of the Monte Carlo estimate from the analysis.

Figure 4.9 a shows the superposition learning processes for PPO-1L, PPO-2L, and PPO-3L. The parameters used for the experiments PPO-1L, PPO-2L, and PPO-3L are delineated in the table 4.3. The learning plots are drawn as the average along with the range of values for three distinct seed values. The parameter \mathbf{M} , for PPO-1L, PPO-2L, and PPO-3L, is calculated from equation 4.8 for the RMS value $\epsilon^2 = 7.8 \times 10^{-3}$ where the values of V_l and C_l are taken from the analysis mentioned above. In other words, we compare the results among PPO-1L, PPO-2L, and PPO-3L for a constant RMS accuracy. Note that these choices of values are done only in

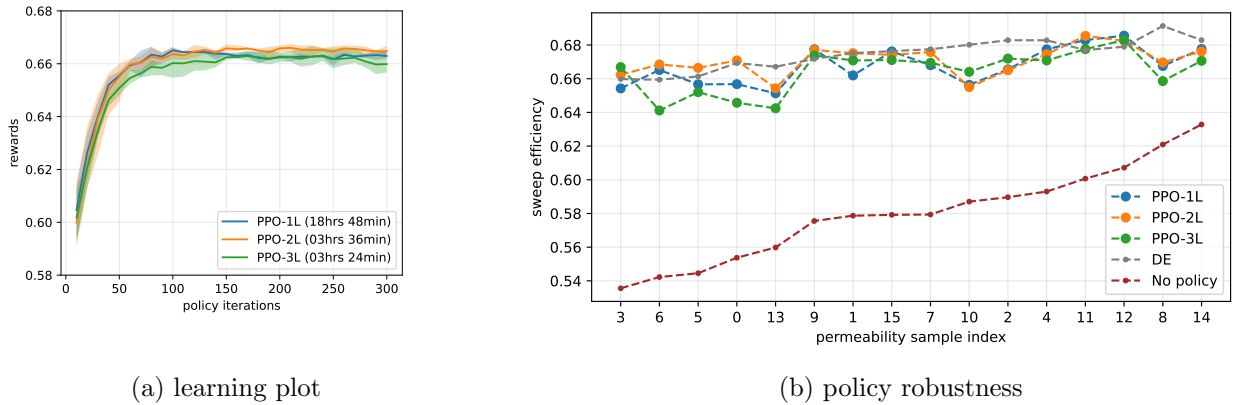


Figure 4.9: multilevel PPO results for ResSim-v1

Table 4.3: parameters of multilevel PPO experiment for ResSim-v1

| | T | M | N | K |
|--------|-------------|---------------|-----|-----|
| PPO-1L | {50} | {250} | 50 | 20 |
| PPO-2L | {70, 5} | {350, 25} | 50 | 20 |
| PPO-3L | {80, 10, 5} | {400, 50, 25} | 50 | 20 |

order to demonstrate a fair comparison among PPO-1L, PPO-2L, and PPO-3L. In practice, it is not required to perform the analysis in order to choose M . Other parameters of the algorithm are tuned to find the convergence for the PPO-3L case first, and these same parameters were used in the PPO-2L and PPO-1L cases. Figure 4.9 a shows the evaluation of the environment policy corresponding to the target task. This policy evaluation is represented with the average reward corresponding to all the permeability samples used in the learning process. PPO-1L refers to the classical PPO algorithm, which takes around 19 wall clock hours, while PPO-2L and PPO-3L which correspond to the proposed multilevel PPO algorithm achieve the same learning in about three and half hours. In other words, we save around 82% computational costs with the proposed algorithm compared to its classical counterpart. Figure 4.9b shows the robustness of the learned policies against uncertainty in permeability. This is done by plotting rewards for 16 random permeability samples of the uncertainty distribution that were unseen during the learning process. These results were compared with the optimal solutions obtained using the differential evolution algorithm (implemented using the SciPy library, [67]), which are denoted DE in Figure 4.9 b. The algorithm parameters for the PPO and DE algorithms, in this study, are delineated in the Appendix C.5.

4.5.2 ResSim-v2 results

Similarly to ResSim-v1, we perform multilevel PPO analysis with PPO-1L, which consists of a total of 1200 policy iterations and is performed every 120 iterations. Figure 4.10 illustrates the correlation between Monte Carlo and the multilevel Monte Carlo estimate of the objective function. The RMS values in ϵ are set to $\{\sqrt{10^{-2}}, \sqrt{10^{-3}}, \sqrt{10^{-4}}\}$ in the analysis. The cost

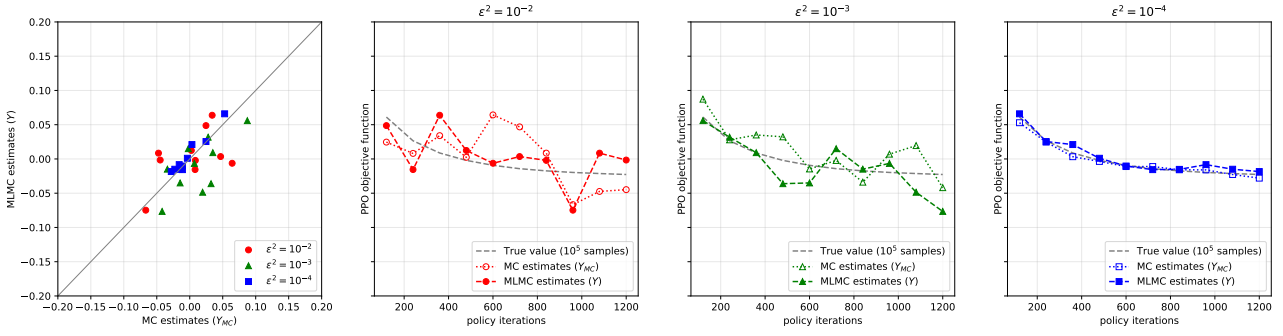
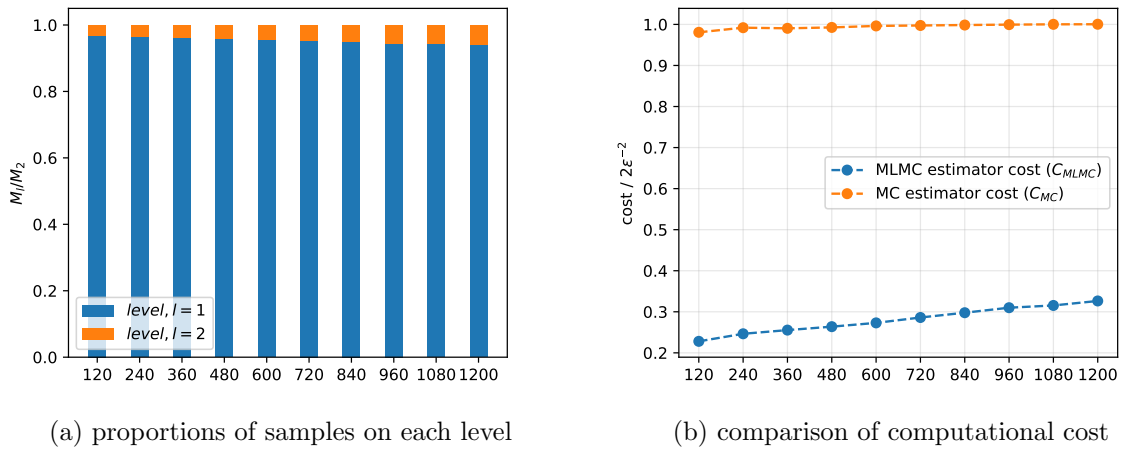


Figure 4.10: comparison of Monte Carlo and multilevel Monte Carlo estimate of PPO objective function for ResSim-v2



(a) proportions of samples on each level

(b) comparison of computational cost

Figure 4.11: MLMC analysis results for ResSim-v2

terms $\{C_1, C_2\}$, which correspond to the computational cost of each term in the multilevel Monte Carlo estimate, are set to $\{0.24, 1.24\}$. As illustrated in Figure 4.10, we observe a higher correlation between Monte Carlo and multilevel Monte Carlo estimates for higher RMS accuracy values. Figure 4.11 a illustrates the proportions of the optimal number of samples at both levels of the multilevel estimator. The comparison between the computational cost of Monte Carlo and the multilevel Monte Carlo estimate is plotted in figure 4.11b. We observe that the computational cost of the multilevel estimate takes only around 25 to 35% of the Monte Carlo estimate from the analysis.

Figure 4.12 a shows the comparison between the learning processes for PPO-1L and PPO-2L. The parameters used for the PPO-1L and PPO-2L experiments are delineated in the table 4.4. In this case, the comparison between PPO-1L and PPO-2L is made for the RMS accuracy

Table 4.4: parameters of multilevel PPO experiment for ResSim-v1

| | T | M | N | K |
|--------|-----------|-----------|-----|-----|
| PPO-1L | {100} | {500} | 50 | 20 |
| PPO-2L | {140, 15} | {700, 75} | 50 | 20 |

value $\epsilon^2 = 3.9 \times 10^{-3}$. Similarly to the ResSim-v1 case, the hyperparameters of the algorithm

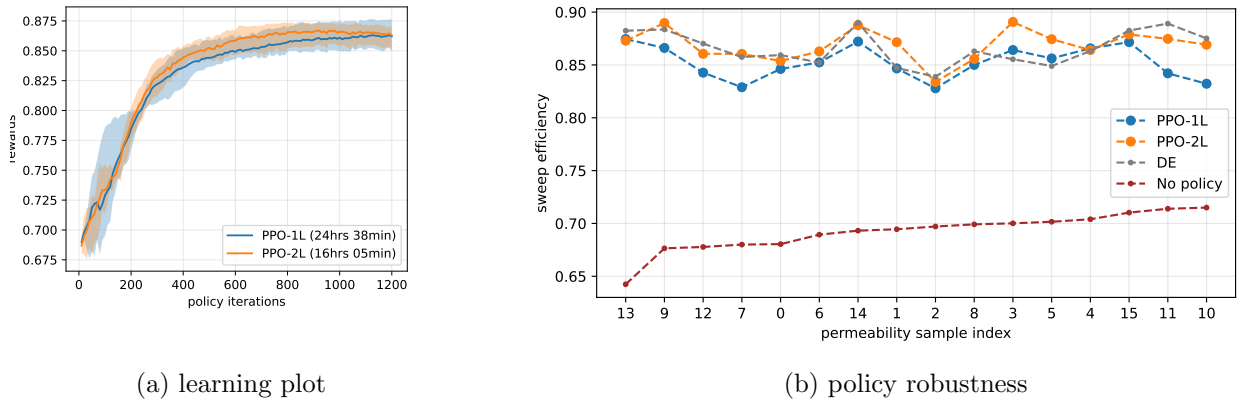


Figure 4.12: multilevel PPO results for ResSim-v2

are tuned to find convergence for the PPO-2L case, and the same parameters were used in the PPO-1L case. Figure 4.9 a shows the average evaluation of the environment policy corresponding to the target task (level 2). PPO-1L refers to the classical PPO algorithm which takes around 24 wall clock hours, while PPO-2L, which corresponds to the proposed multilevel PPO algorithm, achieves the same learning in about 16 hours. In other words, we save around 35% computational costs with the proposed algorithm compared to its classical counterpart. Figure 4.12b shows the robustness of the learned policies against uncertainty in permeability.

4.5.3 challenges and further research direction

Albeit successful results in learning and analysis of the proposed framework, we believe that this study deserves deeper mathematical investigation and analysis. In particular, we would like to study and analyze the effect of the approximation introduced in the MLMC estimation. As an introduction to the proposed framework, the experiments presented for multilevel PPO were performed for a specific PDE-based control problem for flow through porous media. Subsequently, we would like to provide a thorough study with a variety of experiments with the proposed multilevel PPO algorithm. This study would be mainly aimed at general benchmark problems in which RL is utilized to achieve superhuman controls, but with excessive computational costs. Furthermore, while tuning the algorithm parameters for the multilevel PPO algorithm, we observed that increasing the number of levels, the learning rate, and the clip range had an adverse effect on the learning convergence.

4.6 Conclusions

A multilevel framework for deep reinforcement learning is introduced in which the learned agent interacts with multiple levels of PDE-based environments where the level corresponds to the grid fidelity of PDE discretization. We present a mathematical framework that allows the synchronized implementation of task trajectories at multiple environmental levels. The

presented approximate MLMC estimate is at the heart of the proposed multilevel framework. We also present a novel multilevel variant of the classical PPO algorithm based on the proposed multilevel framework. The computational efficiency of this multilevel PPO algorithm is illustrated for two environments for which model dynamics is represented with PDEs describing an incompressible single-phase fluid flow through a porous medium. We observe substantial computational savings in the case studies presented (approximately 82% and 35%, respectively).

As a future scope of this study, we aim to analyze the effect of the presented approximation to standard MLMC estimation. For the multilevel PPO algorithm, this can be done by extending the analysis methodology (presented in Section 4.3.2) for the approximate MLMC estimate. We also aim to provide a future study to benchmark multilevel PPO algorithm performance on a variety of environments.

Chapter 5

Summarized Outlook of Proposed RL Frameworks for Subsurface Flow Control

The contribution of the presented research work can be summarized as a proposal of mathematical frameworks that can be used as stencils for the application of reinforcement learning (RL) in subsurface control. We begin in Chapter 2, by introducing an RL framework to solve the stochastic optimal well control problem for partially observable simulation data. Furthermore, after acknowledging the computational intractability of this framework, we propose an explicit (Chapter 3) and an implicit (Chapter 4) RL framework to alleviate the overall computational cost in the learning process. In this chapter, we summarize the proposed frameworks in order to give an outlook in the context of RL application in reservoir management. We start by comparing the RL-based approach with the conventional closed-loop reservoir management (CLRM) approach, where we highlight the advantages and disadvantages of both approaches. In addition, we provide a brief summary of the proposed frameworks and their methodical connection to each other. We provide a chronological story of these frameworks in relation to each other. This illustrates the general evolution of the RL frameworks presented. Finally, we provide a general set of guidelines in terms of concerns regarding the application of these frameworks to full-field models. We present current state-of-the-art research in RL-based subsurface flow control to give a perspective on the future of RL in reservoir management. Finally, we briefly discuss the application of the proposed frameworks in future research on reservoir management research.

5.1 Introduction

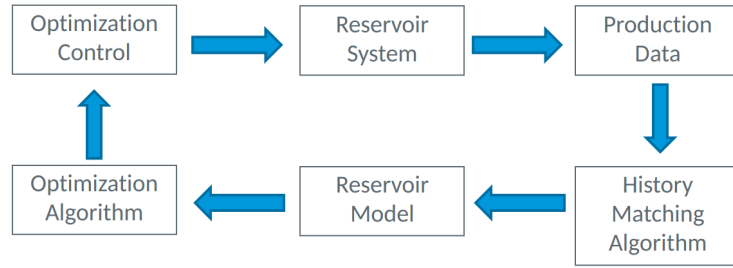
Conventionally, most state-of-the-art research methods for solving optimal well-control problems make use of classical optimization algorithms. This research methodology is classified as closed-loop reservoir management (CLRM) in the field of petroleum engineering. The research work presented in this thesis proposes a paradigm shift in solving the optimal well control problem. Reinforcement learning research, being in its infant form, has yet to see more improvement in its modeling and application to real-world control problems. Naturally, we anticipate that more and more research in RL-based optimal well control will be conducted in the coming years. Let us begin by understanding the fundamental differences between CLRM-based and RL-based approaches to solving optimal well control problems.

5.1.1 CLRM-based approach

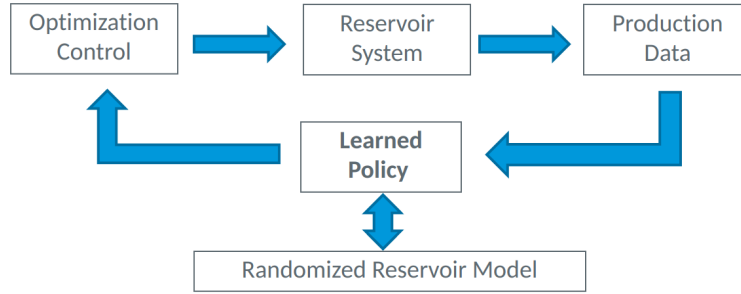
Subsurface flow control problem in reservoir management often consists of two main challenges. First, reservoir simulations are modeled with parameters that are often unknown due to the sparsity of available field data. As a result, these model parameters are often treated as uncertain. This uncertainty can be described using a posterior probability distribution based on prior information in available field data. Second, throughout the reservoir lifecycle, data are only available at certain locations (e.g., well locations). CLRM-based approach to solving the subsurface flow control problem is outlined in Figure 5.1a. In this approach, an initial reservoir model is developed using the initial assumptions in the model parameters. At every control step throughout the reservoir lifecycle, new production data is recovered. This data is used as a prior in history matching algorithms to modify the posterior distribution of uncertain model parameters. The updated reservoir model with these calibrated model parameters is further used in an optimization algorithm. The optimal controls, thus obtained, are then used for the corresponding control step. In order to handle the uncertainty in the model parameters, common practice involves forming an average of objective functions corresponding to certain model realizations. In a nutshell, the CLRM-based approach consists of two fundamental steps at every control step in the reservoir lifecycle: History matching and optimization. Both history matching and control optimizations are mathematically formulated as optimization problems. Numerous research studies are conducted to propose various optimization algorithms for these two steps.

5.1.2 RL-based approach

The approach presented in this thesis offers a paradigm shift from the conventional CLRM-based approach. The aim is to build a policy that can map real-time production data to optimal well controls. With such a policy, we avoid the two steps of history matching and optimization. We propose using reinforcement learning methods to learn this policy using a pre-defined reservoir model. The uncertainty in the reservoir model parameters is considered using a cluster-based domain randomization technique while training the policy. The learned



(a) Conventional CLRM-based approach



(b) Proposed RL-based approach

Figure 5.1: Approaches to solving optimal well control problems

policies can be represented with a deep neural network. As a result, we can employ various deep reinforcement learning algorithms that can be suitable for continuous control variables.

5.1.3 Advantages of RL-based approach

Model uncertainty is one of the most pressing issues in optimal well control problems. We only have access to some sort of uncertainty distribution of the model parameters rather than the true values. As a result, we should find the optimal controls that are conditioned on the uncertainty of the model parameter. For this reason, it makes sense that our solution to optimal well control problems should also form some sort of conditional formulation. Since the RL-based approach offers a conditional policy (where controls are conditioned on the production data), it is in principle an apt method to solve the robust optimal well control problem. This is in contrast to the optimization approach, where the solution corresponds to deterministic values of the optimal controls rather than a conditional form. This is because conventional optimization algorithms typically do not contain a mechanism to handle the uncertainty in the objective function. Some researchers deal with this by treating the objective function as an expectation of a number of model realizations. The optimal controls thus obtained are found by giving equal importance to all the model realizations under consideration rather than being conditioned on the model uncertainty. Note that our aim is to find controls that can be optimal for any of the model realizations. In the above-mentioned approach, we instead find the optimal controls that are applicable to an average of model realizations. Furthermore, history matching is also a complicated and computationally intensive step in the CLRM-based approach. With the RL-based approach, this step can be omitted by implementing robust domain randomization

methods while learning the optimal policy.

5.2 A summarized outlook of proposed RL frameworks

The advantages of the RL-based approach over the CLRM-based approach make RL a lucrative technique to solve optimal well control problems for reservoir management. Since the research study provided in the second chapter, there have been several research studies that implemented subsurface flow control problems using reinforcement learning [62, 63, 75–80]. For the RL-based approach to replace conventional CLRM-based approaches, it must address two key issues in subsurface flow control problems. First, the optimal policy should be able to respond to the partially available reservoir data. Second, the policy should address the uncertainty in the parameters of the reservoir model. Miftakhov, Al-Qasim, and Efremov [62], He et al. [78], and Zhang et al. [79] present useful research studies; however, the policy representation is for the full reservoir data rather than its partial representation. Furthermore, [62, 75, 76] are some examples of research studies in which the RL-based approach is presented for fixed reservoir models (that is, without considering uncertain model parameters). The research studies done in [63, 64, 80] provide great examples of RL-based approaches which take into account both of these issues. As RL-based approaches are being explored in reservoir management research studies, we must also address one of its main bottlenecks. Since the RL-based approach requires a high number of reservoir simulation runs, it is computationally demanding. So far, we have not seen much state-of-the-art research that has been done to specifically address this issue in reservoir management research studies. At the end of the research study in Chapter 2, we found ourselves at a crossroads either to further our studies by applying the proposed RL framework on real-life reservoir management problems or to improve the proposed RL framework itself to make it more computationally efficient. We chose to further our research in the latter direction, since the computational cost is the most pressing issue for simulation-based RL problems. As a result, with the computationally efficient solution provided in this research, we not only help improve RL-based approaches in reservoir management but can also address this issue for other simulation-based problems. We present two distinct approaches to improve the computational efficiency of RL-based approaches. The first approach makes use of transfer learning techniques by using multiple grids for simulation. Next, we propose a more generalized reinforcement learning framework that employs the multilevel Monte Carlo estimation method.

As we look back at the structure of the proposed RL frameworks in this thesis, we cannot help but observe a thematic progression in the frameworks presented. Figure 5.2 illustrates the evolution in the presented frameworks in terms of the outline of the agent-environment interaction. The RL framework presented in Chapter 2 simply follows the classical agent-environment (illustrated in figure 5.2a). In this case, the environment is designed by instilling cluster-based domain randomization to account for uncertainty in the environment. In the third chapter, we present a multigrid framework that employs a transfer learning technique. In this framework, the policy is designed for the high-dimensional state and action space (corresponding

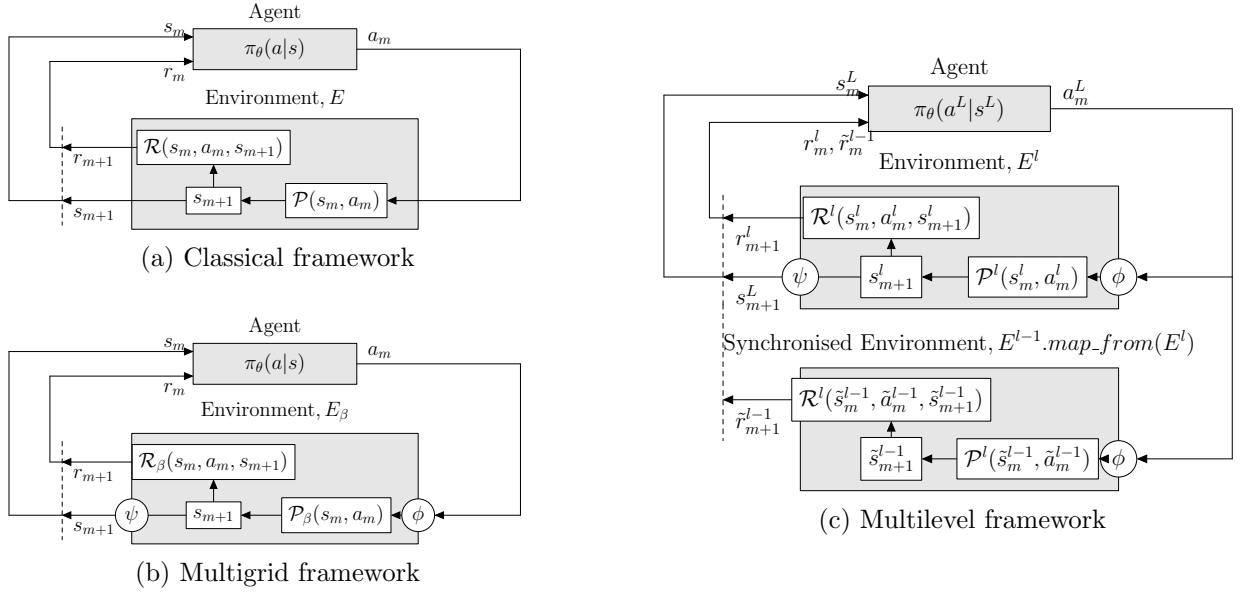


Figure 5.2: Evolution of proposed RL frameworks

to the finest grid). The environment is parameterized by the grid fidelity factor β that is changed progressively through the policy learning process. As a result, we introduce the mappings ψ and ϕ that map the state and actions from the dimensions of the environment's state and action space to a high-dimensional state and action space corresponding to the policy network. Figure 5.2b illustrates the multigrid agent-environment where the environment is indicated by E_β and the mapping functions to and from the environment are denoted by ψ and ϕ .

The multilevel framework presented in Chapter 4 can be viewed as a progression to the multigrid framework presented in Chapter 3. Like the multigrid framework, the policy network in a multilevel framework is also designed for high-fidelity state-action space. As a result, similar mapping functions like ψ and ϕ are used in the multilevel framework. Furthermore, multiple levels of the environment in the presented studies are represented by the coarseness of the grid and are indicated with a superscript with its level number l . The key idea in the multilevel framework is to use multilevel Monte Carlo (MLMC) estimates of the objective function in RL algorithms. To formulate the MLMC estimate of the objective function, we propose a synchronized implementation framework in which each environment on level l is synchronized with the environment on level $l - 1$. Figure 5.2c outlines the agent-environment interaction for the proposed multilevel framework.

The multilevel and multigrid frameworks can be seen as independent approaches to improve the computational efficiency of the RL framework presented in Chapter 2. Although these frameworks are distinct in their approach, it begs the obvious thought to compare these frameworks in relation to each other. The multigrid framework is conceptually based on the principles of transfer learning. On the other hand, the multilevel framework is based on a novel paradigm in reinforcement learning, which is introduced in this thesis. Despite the successful results presented in Chapter 4, we should note that this research is in its infancy stage and deserves

more refinement in related future research work. One way to compare these frameworks would be to study the hyperparameters introduced in the algorithms presented. The multigrid PPO algorithm must be tuned with parameters such as δ , n , β and \mathbf{E} (refer algorithm 4 for details). Furthermore, the environment in these algorithms must be provided with mapping functions (similar to ϕ and ψ in figure 5.2b). On the other hand, the multilevel PPO algorithm (refer to algorithm 7) can be viewed as simply the vectorization of the classical PPO algorithm. In other words, we do not have to consider any extra hyperparameters like in the multigrid PPO algorithm. For the multilevel PPO algorithm, we instead provide vectorized inputs (e.g. \mathbf{T} , \mathbf{M} and \mathbf{E} from algorithm 7) corresponding to the levels of the environment under consideration. In addition to the mapping functions provided in the multigrid framework, environments in multilevel frameworks are also needed to provide the mapping capability to implement synchronization of multiple levels.

It will also be important to compare the multigrid and multilevel frameworks for their computational efficiency. This can be done by comparing the computational savings from these frameworks using a benchmark problem. In this thesis, we presented both frameworks as independent research studies. As a result, the case studies chosen in these research studies (albeit similar) are not identical. For instance, the grid fidelity factors in the multigrid framework and multiple levels in the multilevel framework are different from each other. For this reason, it would not be fair to compare the computational effectiveness of these frameworks with the case studies presented. As a future research direction, it would be interesting to design benchmark case studies to compare the multigrid and multilevel frameworks.

5.3 Guidelines for application of proposed framework

The main contribution of this thesis is in terms of mathematical frameworks that can be used to solve optimal well control problems in reservoir management. The effectiveness of these frameworks is demonstrated by applying them on prototypes of some reservoir simulation problems that represent key features in a typical reservoir simulation. A natural progression of these research studies is to apply these mathematical frameworks in real-life reservoir simulation problems. The research studies presented in [62–64, 75, 76, 79–82] form an initial building block for implementing RL-based approaches in optimal well control problems. The general guidelines for reservoir engineers to use the RL-based approach in optimal well control problems can be delineated as given below.

Policy network: The policy in optimal well control problems maps a certain state of reservoir simulation to the optimal controls. This policy can only be applicable in real-time applications if the state is represented in the form of on-field production data (e.g., data at well location). Some research studies (e.g., [76, 83]) formulated the policy in terms of a tabular Q-value by assuming discrete state-action space. However, in reality the real-time field data is often of the continuous type. To accommodate this, research studies like [79] approximated Q-value function as an artificial neural network (ANN).

However, in this study the action is presented in a discrete format, where the controls refer to a delta increment or decrement. For an accurate representation of the control of the field wells, we recommend a continuous representation in Chapter 2, where the policy network is represented as an ANN. For the simplified prototype presented in this study we found that ANN representation for the policy is sufficient. However, research studies presented in [63, 80] recommend a recurrent neural network called transformer. Taking into account the POMDP (partially observable Markov decision process) nature of the optimal well control problem, the recurrent neural network makes a suitable candidate for the policy network in the RL-based approach.

Action representation: Research studies on RL-based optimal well control problems can be divided into two categories as far as the representation of actions (well controls). Research studies [62, 76] represent the injection/production rate of the well in a discrete format. Injection / production rates are controlled as delta increment, decrement, or no change. However, this does not necessarily reflect the reality of on-field wells, which are controlled through the specification of bottom-hole pressures (BHPs). On the other hand, studies like [79, 80, 83] represent well controls more accurately in a continuous form. The action representation in this case is a vector representing the flow controls at each wells. In order to use the RL framework from Chapter 2 in real-life reservoir problems, we recommend representing the well controls in terms of BHPs at production wells and injection flow rates at injection wells.

State representation: Almost all research studies identify saturation and pressure values as the representation of the state in RL-based approaches. Studies like [41, 62] are an important contribution to RL-based approaches to optimal well control where states are represented from all locations on the grid. However, they cannot be applied in real-time reservoirs where the state can only be observed at certain well locations. Hourfar et al. [75], Zhang et al. [79], and Nasir and Durlofsky [80] represent the state only at observed well locations. It is important to consider the POMDP nature of the optimal well control problem. As a result, the state is supposed to be represented with an array of observations for a buffer of observations in certain past control steps.

Reward formulation: To facilitate implementation and representation, we chose sweep efficiency as a reward for RL-based approaches in the studies presented in this thesis. For real-life applications on subsurface flow control problems, reservoir engineers may want to choose the cumulative net present value (NPV) as the reward. The reward is calculated for every control step in the policy learning process.

Domain randomization: Very few RL-based research studies consider domain randomization while learning the optimal policy. To address the uncertainty of the model in reservoir simulations, domain randomization plays an important role. In Chapter 2, we describe a cluster-based analysis methodology to explain the domain randomization step in RL-based approaches. With such an approach, reservoir engineers can assess the

robustness of the learned policy in real time. In fact, we recommend training the optimal control policy based on the robustness in such evaluation. In order to consider the dynamic differences between various model realizations, we advise reservoir engineers to use the connectivity distance metric [84].

Multigrid framework: Since the research in RL-based approaches in optimal well control is still in its infant stage, we have not seen many research studies focusing on improving the computational efficiency of RL frameworks. As a result, the research study presented in Chapter 3 marks the first step toward implementing a computationally efficient RL framework for optimal well control problems. In order to accommodate the transfer learning between multiple grids of the environment, we recommend using mapping functions similar to the ones explained in Chapter 3 (please refer to section 3.2). Note that these mapping functions will be computed for every agent-environment interaction (i.e. each control step). As a result, reservoir engineers need to ensure that these operations are computationally optimized to avoid overhead computational costs. In our research study, we programmed the mapping functions with careful computational optimization. In addition, we used a Python library called NUMBA [85] to convert our functions into fast machine codes. The selection of grid fidelity factors β is delineated in section 3.3. Additionally, hyperparameters such as δ and n can be tuned on a case-by-case basis. However, from a practitioner’s perspective, we advise using the δ value in the range 0.1-0.7 of and n from the range 5-15.

Multilevel framework: In an attempt to provide a generalized reinforcement learning framework that can alleviate the computational cost in RL-based approaches for simulation-based control problems, we present the multilevel framework in Chapter 4. Despite the success in the numerical experiment presented, we acknowledge that this framework still needs more in-depth mathematical investigation. One of the major limitations that we observed in this framework was about the stability of the proposed multilevel PPO algorithm when we increased the number of levels of environments. We recommend that reservoir engineers employ at most three levels of environments in this framework. As a rule of thumb, we recommend using a very high number of time steps at the coarsest level (first level) of the environment. For the rest of the levels, the number of timesteps is chosen in a decrementing order. These parameters can be tuned on a case-by-case basis.

5.4 Future scope

We present numerical experiments using single-phase models for the proposed frameworks. We discuss the choice of single-phase models over two-phase models for the ease of execution and demonstration in section 2.2. Having said that, we carefully designed these case studies to represent subsurface control problems with real full-field models. We assume the uncertainty of the model by treating the permeability as the uncertain model parameter. In real full-field

models, other model parameters might as well be subject to uncertainty. However, we chose a single parameter (i.e. permeability) for simplicity in representation. Note that the methodology presented in these frameworks would nonetheless be similar for more than one uncertain model parameter. Furthermore, it is also important to select the uncertainty distribution with careful consideration for it to be significant in subsurface flow control problems. Sometimes, uncertainties in the model parameter might not reflect direct uncertainty in the optimal controls. As a result, we chose the uncertainty distribution such that its uncertainty is directly reflected in the optimal policies. For instance, the change in the position of the high-permeability channel in the presented case study (case 1 in Chapters 2 to 4) directly reflects the change in the optimal policy of restricting the flow through wells near the channel. Furthermore, we present the values of the model parameters in a range similar to that presented in the standard SPE-10 model 2 [51] (detailed in 2.3).

A natural extension to the work presented would be to apply the presented RL frameworks in full-field reservoir models. In this section, we discuss some key concepts that should be considered when applying this work to such problems. For ease of implementation, we represent the uncertainty of the model parameter only in terms of permeability. However, in reality, we might have other model parameters (e.g. porosity, irreducible saturation values, etc.) which might also attribute some uncertainty. In the framework presented, the uncertainty in the subsurface controls was dealt with using a cluster-based analysis. The same cluster-based methodology can also be extended to cases where more than one parameter is uncertain given the predefined uncertainty distribution of these parameters. This is possible because we are using the connectivity distance metric while clustering, which measures the distance between dynamic outputs of the model realizations rather than the distance between model parameters themselves. Furthermore, we assume that the flow is incompressible in the case studies presented. To consider compressible flow in the full-field models, we need to rethink the representation of action in the proposed RL models. In the proposed methodology, we present an action representation using a weight vector that represents the flow proportion through each production well (detailed in Section 2.2). For the compressible flow scenario, it would be more accurate to represent the actions in terms of BHPs at production well locations (similar to that in the research study [80]).

Computational cost is the main concern in the use of RL-based approaches in subsurface controls. Let us consider the timescale for application of the proposed framework in full-field reservoir benchmark models such as PUNQ-S3. The average simulation time for this problem is around 10 minutes. As observed in Chapter 3, we assume the policy learning process to require around 60000 simulation runs using 64 CPUs. For the PUNQ-S3 case, this would mean that it would take around six and a half days to finish the learning process. Note that in the presented studies, we are using a desktop with 64 CPUs (Intel Xeon CPU E5-2699 v3 @ 2.30GHz) and a GPU (NVIDIA, GeForce RTX 2080 Ti) to implement this experimentation. So far, for instance, if we instead use 256 CPUs, this time can be reduced to just one and a half days. Furthermore, to demonstrate the complete learning process, we illustrated the learning

process for 60000 simulation runs. However, as we can see, we reach an optimal policy just before 40000 simulation runs, which would correspond to four days for the PUNQ-S3 case. Having said that, let us consider the scenarios presented in Chapter 2 just as they are for the rest of the discussion. Further, we observe around 60-70% computational savings with the multigrid framework presented in Chapter 3. That is, we would expect to reduce the policy learning time from 6.5 to around 2 days when using the multigrid framework with the same computational resources. In addition, we also anticipate observing similar computational savings using the multilevel PPO algorithm.

5.5 Conclusion

This chapter provides a summary of the proposed RL-based optimal control research along with its future scope. We distinguish RL-based approaches from the conventional CLRM-based approach and discuss its advantages. In addition, we provide all the RL frameworks proposed in this thesis and how they are related to each other. We supplement this summary with state-of-the-art research in RL-based research applications in reservoir management. On the basis of the proposed RL frameworks and state-of-the-art research, we delineate guidelines for reservoir engineers regarding RL-based approaches in subsurface control problems. Finally, we share our perspective on the future scope of applications of proposed RL frameworks in full-field reservoir models.

Chapter 6

Conclusion and Future Work

In this thesis, we study the application of reinforcement learning to solve subsurface flow control problems. We introduced an RL framework to solve the stochastic optimal well control problem for partially observable simulation data (Chapter 2). After acknowledging the computational cost used by simulation runs as the major limitation of the proposed RL algorithm, we propose an explicit (Chapter 3) and an implicit (Chapter 4) RL approach to alleviate the overall computational cost of the proposed RL framework. In the following, we recapitulate the main findings of our study along with possible extensions.

6.1 Chapter 2: Stochastic optimal well control in subsurface reservoirs using reinforcement learning

The framework is presented to use model-free RL algorithms to obtain a robust optimal control strategy for optimal well control problems using the Model-Free RL algorithm. This policy is learned assuming that the system is partially observable and governed by a nonlinear partial differential equation system. The robustness of these policies was achieved using a domain randomization scheme that uses only a few samples from a predefined uncertainty distribution using cluster analysis. Furthermore, the optimality of these policies has been successfully compared to reference solutions obtained by directly optimizing using the differential evolution algorithm. We see the current framework as a first attempt to apply narrow AI to the subsurface flow control field, in which data are only available at well locations. In the current study, we trained RL policies with a large number of simulation runs. This is computationally expensive in large-scale models with large simulation run-times. Finally, we discuss a possible resolution of this issue by considering fast surrogate modeling techniques to accelerate the reinforcement learning process.

6.2 Chapter 3: Robust optimal well control using an adaptive multigrid reinforcement learning framework

We have introduced an adaptive multigrid multigrid RL framework to solve a robust well control optimal problem. The proposed framework is designed to be general enough to be applicable to similar optimal control problems governed by a set of time-dependent nonlinear PDEs. Numerically, we observed a significant reduction in policy learning computational costs compared to the results of the classical PPO algorithm. In the presented case study, computational savings of 61 % in simulation run time were observed in test case 1 and 76 % in test case 2. However, we noted that these results are highly dependent on the right choice of the algorithm hyperparameters (denoted with δ , n , β and \mathbf{E} in the chapter) which were tuned heuristically. As a future direction for this research study, we pointed to a future direction in the analysis of optimal values for β that maximizes overall computational savings. Additionally, policy transfer was performed sequentially in the proposed framework, which appeared to work optimally. However, it is important to examine the impact of the sequence of policy transfer on overall performance to improve the generality of the proposed framework.

6.3 Chapter 4: Multilevel framework for deep reinforcement learning

We present a multilevel framework for deep reinforcement learning algorithms by introducing synchronized rollouts at multiple levels of predefined environments. We present this framework by introducing the multilevel variant of the PPO algorithm. The computational efficiency of the multilevel variant of PPO is illustrated for two experiments by comparison with the results of the classical PPO algorithm. We observe substantial computational savings in the case studies presented (approximately 82% and 35%, respectively). Next, our goal is to provide further convergence and stability analysis for the proposed multilevel PPO algorithm.

6.4 Remarks and future directions

In our study, the emphasis was on introducing a general framework for the use of RL in optimal well control problems. The study of the adaptive multigrid framework warrants future investigation of the selection and order of the grid fidelity factor values. In transfer learning research, this is called curriculum learning. We foresee that curriculum learning for the proposed adaptive multigrid framework is a natural next step forward. The introduction of a multilevel framework for reinforcement learning opens the door to many future directions. In the presented study, we demonstrate the multilevel framework using a multilevel variant of the classical PPO algorithm. Similarly, one can create multilevel versions of other state-of-the-art RL algorithms like DQN, A2C, SAC, etc. Research in the direction of selection of levels of environment can also be a valuable addition to the presented multilevel PPO algorithm. This will most likely include

conditions and/or algorithms to choose levels for the environment. More importantly, we would like to further investigate the multilevel PPO algorithm using a diverse set of environments along with a deeper analysis and ablation study. Due to the problem at hand, we presented our results only for PDE-based environments. However, taking the generality of MLMC theory into consideration, we also foresee the application of multilevel PPO in other environments which are not necessarily PDE-based.

The studies provided in this thesis formulate the first step towards the application of narrow AI in reservoir management. The natural next step is to apply the frameworks provided in this thesis to real reservoir case studies. These case studies can highlight the effectiveness of RL control policies compared to current closed-loop reservoir management techniques. The ultimate vision of the proposed research studies is to be able to build a technology to achieve real-time and dynamic optimal schedule for injector/producer wells throughout the reservoir lifecycle. We foresee to achieve this by essentially learning the simulation-based policies, which can be further updated in real time using offline reinforcement learning methods.

Appendix A

Appendices for Chapter 2

A.1 Definition of value and advantage functions

In RL, the policy $\pi(a|s)$ is said to be optimal if it maps the state s_t with an action a_t that correspond to maximum expected return value. These return values are learned through the data obtained in agent-environment interactions. Following are some definition of return values typically used in RL:

Value function is the expected future return for a particular state s_t and is defined as,

$$V(s) = \mathbb{E}_\pi \left[\sum_m \gamma^m r_{m+t+1} | s_t = s \right],$$

where $\mathbb{E}_\pi[\dots]$ denotes expected value given that the agent follows the policy π . As short hand, we denote $V(s)$ at state s_t as V_t .

Q function is similar to value function except that it represent the expected return when the agent takes action a_t in the state s_t . It is defined as,

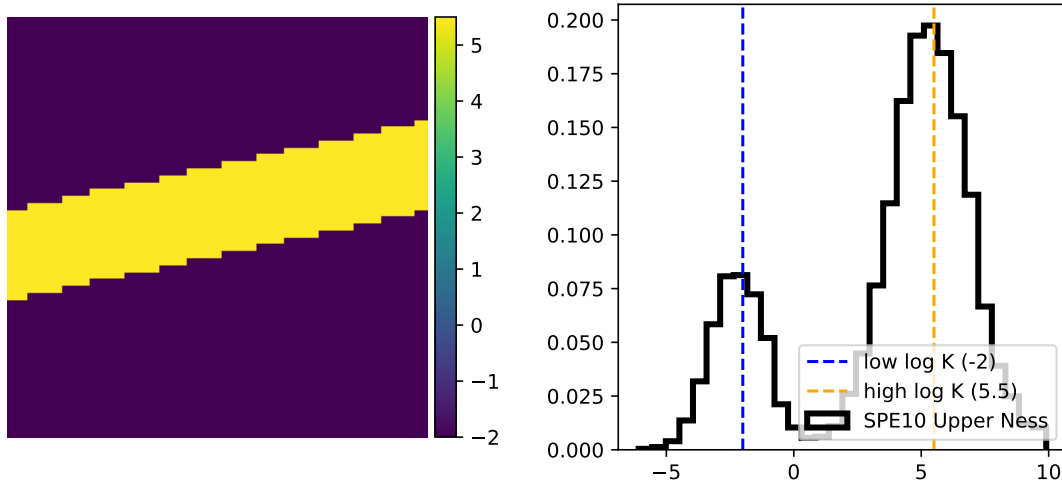
$$Q(s, a) = \mathbb{E}_\pi \left[\sum_m \gamma^m r_{m+t+1} | s_t = s, a_t = a \right].$$

Advantage function is defined as the difference between Q function and value function and is denoted by A_t at state s_t and action a_t .

A.2 Permeability uncertainty distribution parameters

Model parameters of case studies chosen for this paper were inspired from the SPE 10 model 2 parameters. First test case represents channel like permeability distribution which consists of a linear high permeability channel passing through a low permeability domain. The values of high and low permeabilities in this distribution were selected in reference to the Upperness formation

in SPE10 model 2. Figure A.1 shows the plot of Upperness log permeability distribution and an example of log permeability field in test case 1. As can be seen, the high and low log permeability values in test case 1 are chosen from the peaks of Upperness log permeability distribution.



(a) log permeability contour plot (unit: mD) for test case 1

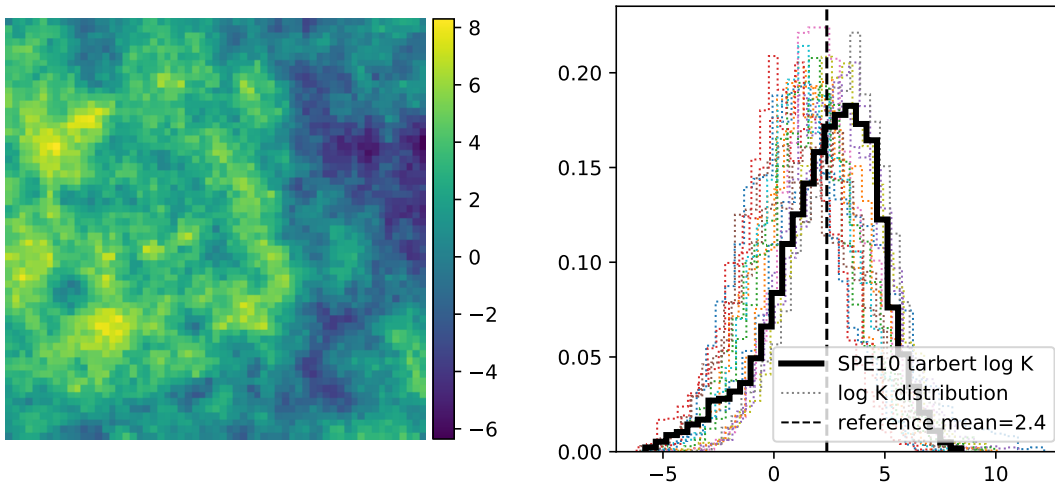
(b) log permeability density distribution for SPE-10 model 2 uppersness case

Figure A.1: Value for low (-2) and high (5.5) log permeability in test case 1 was chosen from the SPE-10 model 2 Upperness permeability distribution peaks

Second test case represents a smoother, spatially correlated permeability distribution often found in geoscience literature. The log permeability distribution is formulated with Equations (2.9), (2.10), (2.11) and (2.12). The correlation length l was selected to be 240 ft in order to consider 20% of the domain height. The idea is to choose a correlation length that is smaller than the quadrants of a domain. The distribution amplitude σ and μ_g is chosen as 2.5 and 2.4, respectively. These values were chosen in order to match the log permeability distribution of test case 2 to match with Tarbert formation from SPE10 model 2. Figure A.2 shows comparison of a test case 2 log permeability field realization along with the superpositioning of test case 2 and Tarbert case permeability distribution.

A.3 RL algorithm parameters

We use stable-baselines 3 library [66] for PPO and A2C algorithms. Parameters used for PPO and A2C are tabulated in table A.1 and A.2, respectively, which were tuned using trial and error. The parameters used to obtain the frozen policy using PPO algorithms are same as those used in PPO parameters presented in these table. The DE algorithm is executed using python's SciPy library [67]. Its parameters are delineated in table A.3. The DE algorithm parameters were tuned by making sure to achieve convergence in optimal results for first four realizations. The same parameters were used for the rest of the 12 realizations. For this reason, we are likely



(a) log permeability contour plot (unit: mD) for test case 2

(b) log permeability density distribution for SPE-10 model 2 tarbert case

Figure A.2: Mean of (2.4) log permeability in test case 2 was chosen from the SPE-10 model 2 Tarbert case data. Log permeability distribution chosen in test case 2 is super-positioned with Tarbert permeability distribution (shown with thick black line) for comparison

to observe some cases where DE algorithms might provide slightly lower optimality as compared to learned optimal policies. For PPO algorithms with full state representation, same parameters (from table A.1) were used except the network MLP layers and learning rates: layers [3721, 4000, 2000, 800, 300, 61] and learning rate $1e-5$ for test case 1 and layers [3721, 4000, 2000, 800, 300, 4] and learning rate $5e-6$ for test case 2. These parameters are tuned in order to obtain the minimum variance in the learning plots. Figure A.3 demonstrate the spread of the learning plot for PPO and A2C for the text case 1 and 2. The code repository for both the test cases presented in this paper can be found on the link: https://github.com/atishdixit16/rl_robust_owc.

Table A.1: PPO algorithm parameters

| | Test case 1 | Test case 2 |
|-------------------------------------|--------------------|-------------|
| number of episodes | 60000 | 60000 |
| number of CPUs, N | 64 | 64 |
| number of steps, T | 50 | 50 |
| mini-batch size, M | 16 | 16 |
| epochs, K | 20 | 20 |
| discount rate, γ | 0.99 | 0.99 |
| clip range, ϵ | 0.1 | 0.1 |
| policy network MLP layers | [93,150,100,80,62] | [9,20,20,4] |
| policy network activation functions | tanh | tanh |
| policy network optimizers | Adam | Adam |
| learning rate | $1e-6$ | $5e-4$ |

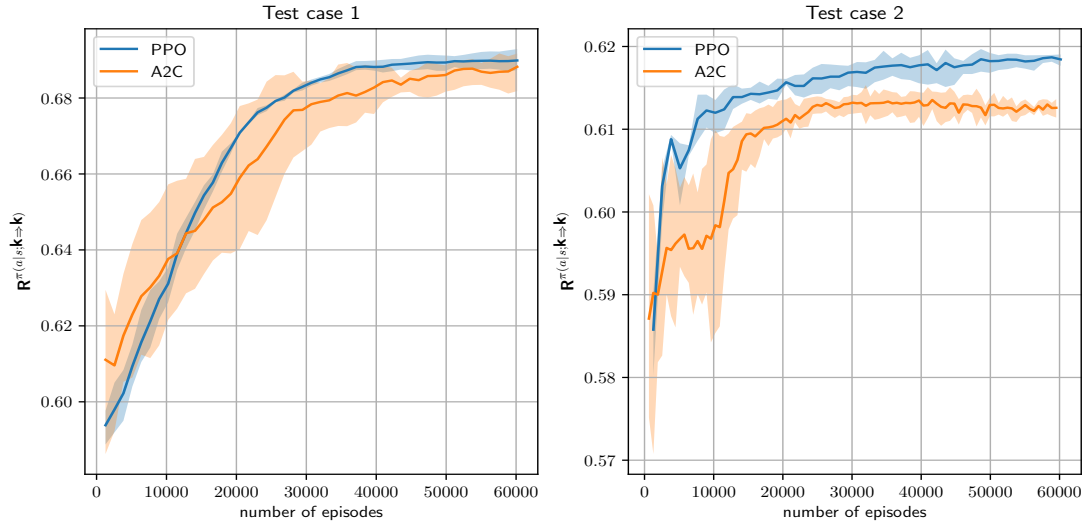


Figure A.3: learning plot range over three distinct seed values for test case 1 and 2

Table A.2: A2C algorithm parameters

| | Test case 1 | Test case 2 |
|-------------------------------------|--------------------|-------------|
| number of episodes | 60000 | 60000 |
| number of CPUs, N | 64 | 32 |
| number of steps, T | 50 | 20 |
| discount rate, γ | 0.99 | 0.99 |
| policy network MLP layers | [93,150,100,80,62] | [9,20,20,4] |
| policy network activation functions | tanh | tanh |
| policy network optimizers | Adam | Adam |
| learning rate | 2e-4 | 1e-4 |

Table A.3: DE algorithm parameters

| | Test case 1 | Test case 2 |
|----------------------|-------------|-------------|
| number of CPUs | 64 | 64 |
| number of iterations | 750 | 750 |
| population size | 310 | 20 |
| recombination factor | 0.9 | 0.9 |
| mutation factor | U(0.5,1) | U(0.5,1) |

Appendix B

Appendices for Chapter 3

B.1 Cluster Analysis of Permeability Uncertainty Distribution

training vector \mathbf{k} is chosen to represent the variability in the permeability distribution \mathcal{K} . For the optimal control problem, our main interest is in the uncertainty in the dynamical response of permeability rather than the uncertainty in permeability itself. As a result, the connectivity distance [84] is used as a measure of the distance between the permeability field samples. The connectivity distance matrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ among the N samples of \mathcal{K} is formulated as

$$\mathbf{D}(k_i, k_j) = \sum_{x''} \int_{t_0}^T [s(x'', t; k_i) - s(x'', t; k_j)]^2 dt,$$

where N corresponds to a large number of samples of uncertainty distribution, $s(x'', t; k_i)$ is saturation at location x'' , and time t , when the permeability is set to k_i and all wells are open equally. Multidimensional scaling of the distance matrix \mathbf{D} is used to produce N two-dimensional coordinates d_1, d_2, \dots, d_N , each representing a permeability sample. The coordinates d_1, d_2, \dots, d_N are obtained such that the distance between d_i and d_j is equivalent to $\mathbf{D}(k_i, k_j)$. In the k-means clustering process, these coordinates are divided into l sets S_1, S_2, \dots, S_l , obtained by solving the optimization problem, defined as

$$\arg \min_S \sum_i^l \sum_{d_j \in S_i} \|d_j - \mu_{S_i}\|,$$

where μ_{S_i} is average of all coordinates in the set S_i . The training vector \mathbf{k} is a set of l samples of \mathcal{K} where each of its value k_i correspond to the one nearest to μ_{S_i} . The total number of samples N and clusters l are chosen to be 1000 and 16 for both uncertainty distributions, \mathcal{G}_1 and \mathcal{G}_2 . A training vector \mathbf{k} is obtained with samples k_1, \dots, k_{16} each corresponding to a cluster center. Figures B.1(a) and B.1(b) show cluster plots for samples of permeability distribution \mathcal{G}_1 and

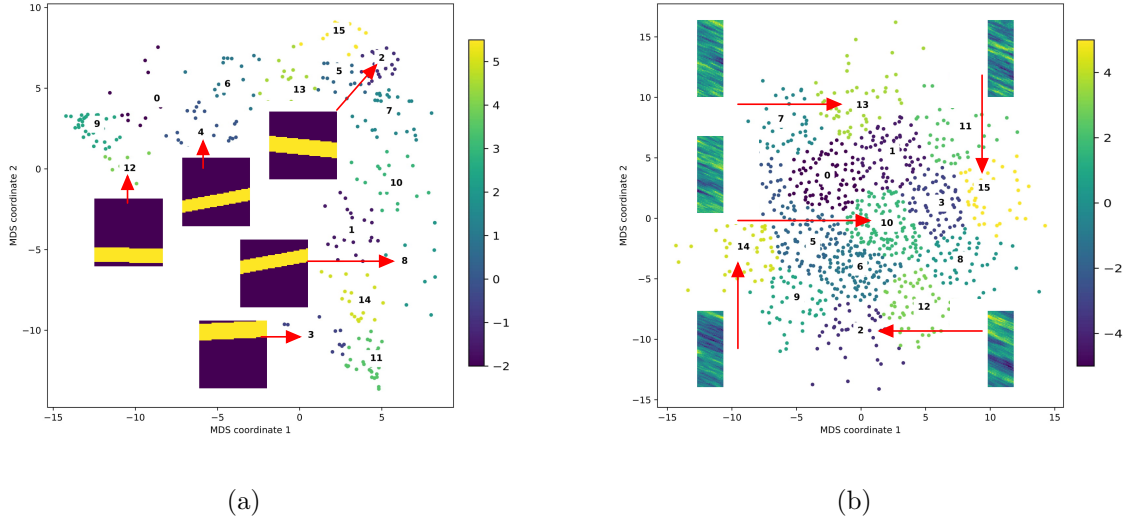


Figure B.1: Log-permeability plots for training data of test case 1 and 2: **a** and **b** illustrate clustering for \mathcal{G}_1 and \mathcal{G}_2 distribution samples

\mathcal{G}_2 , respectively. Furthermore, 16 permeability samples, each randomly chosen from a cluster, are chosen to evaluate the learned policies. Figures 3.9(a) and 3.12(a) illustrate these samples for test case 1 and 2, respectively.

B.2 Definitions of Value and Advantage Function

In RL, the policy $\pi(A|S)$ is said to be optimal if it maps the state S_t with an action A_t that correspond to maximum expected return value. These return values are learned through the data obtained from agent-environment interactions. The following are some definitions of return values typically used in RL:

The value function is the expected future return for a particular state S_t and is defined as

$$V(S) = \mathbb{E}_\pi \left[\sum_m \gamma^m R_{m+t+1} | S_t = S \right],$$

where $\mathbb{E}_\pi[\dots]$ denotes the expected value given that the agent follows the policy π . As a shorthand notation, $V(S)$ in the state S_t is denoted as V_t .

Q function is similar to value function except that it represents the expected return when the agent takes action a_t in the state S_t . It is defined as

$$Q(S, A) = \mathbb{E}_\pi \left[\sum_m \gamma^m R_{m+t+1} | S_t = S, A_t = A \right].$$

Advantage function is defined as the difference between Q function and value function and is denoted by $Adv(S, A)$ at state S and action A .

B.3 Algorithm Parameters

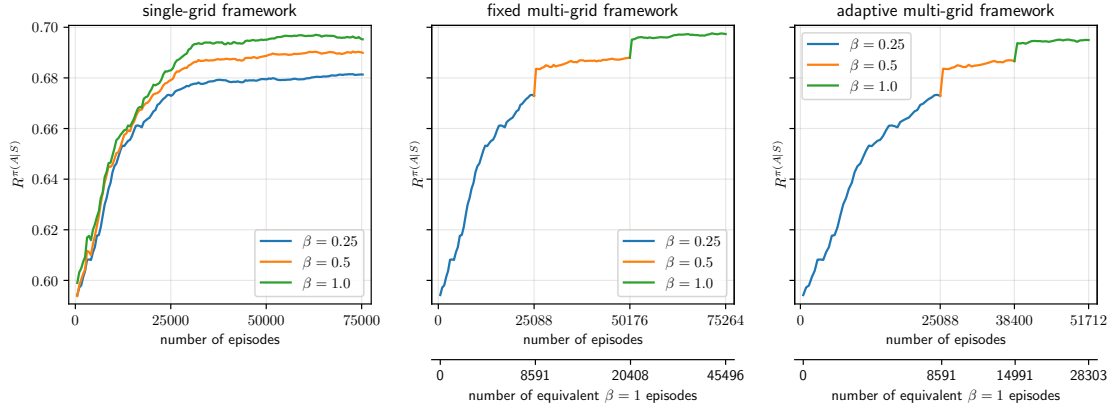
Parameters used for PPO are tabulated in table B.1 which were tuned using trial and error. For the PPO algorithm, the parameters were tuned to find the least variability in the learning plots. Figures B.2 and B.3 show learning plots corresponding to three different seeds to show the stochasticity of the obtained results. The parameters of the DE algorithm are delineated in table B.2. The code repository for both the test cases presented in this paper can be found on the link: https://github.com/atishdixit16/ada_multigrid_ppo.

Table B.1: PPO algorithm parameters

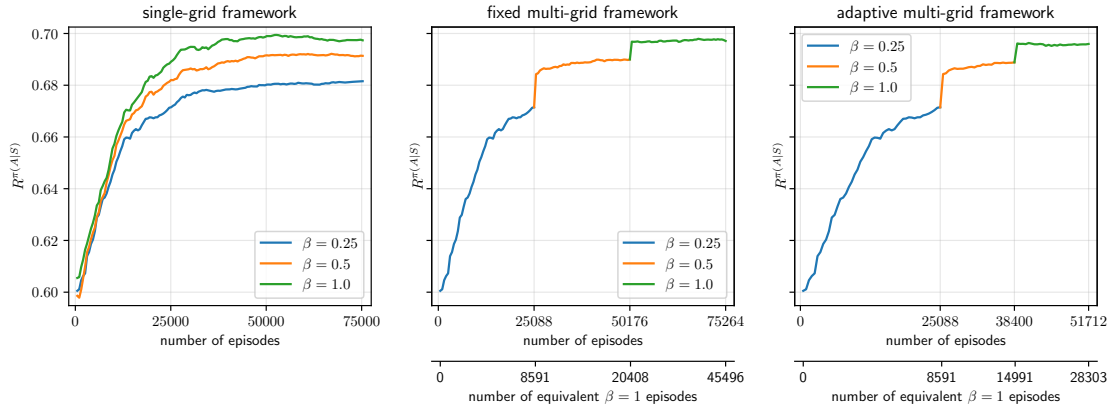
| | case 1 | case 2 |
|-------------------------------------|--------------------|------------------|
| number of CPUs, N | 64 | 64 |
| number of steps, T | 40 | 40 |
| mini-batch size, M | 16 | 16 |
| epochs, K | 20 | 20 |
| discount rate, γ | 0.99 | 0.99 |
| clip range, ϵ | 0.1 | 0.15 |
| policy network MLP layers | [93,150,100,80,62] | [35,70,70,50,21] |
| policy network activation functions | tanh | tanh |
| policy network optimizers | Adam | Adam |
| learning rate | 3e-6 | 1e-4 |

Table B.2: DE algorithm parameters

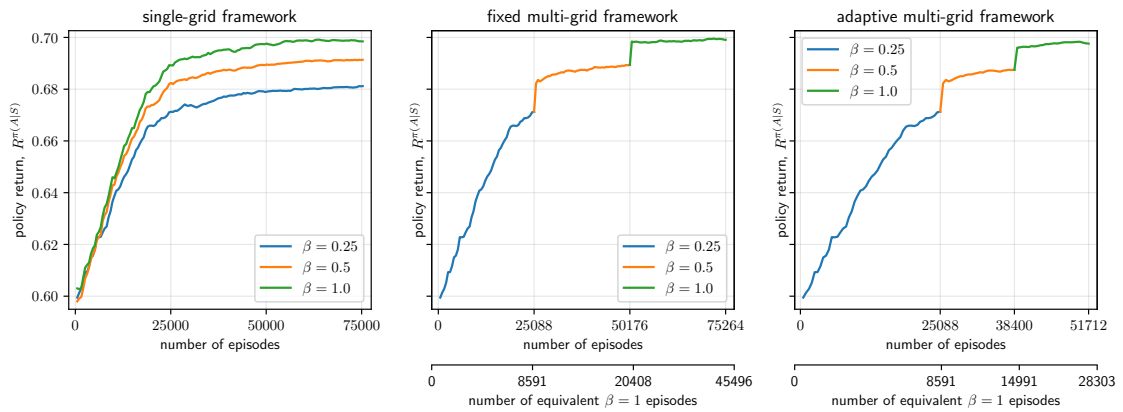
| | case 1 | case 2 |
|----------------------|---------|---------|
| number of CPUs | 64 | 64 |
| number of iterations | 1024 | 1024 |
| population size | 310 | 105 |
| recombination factor | 0.9 | 0.9 |
| mutation factor | (0.5,1) | (0.5,1) |



(a)

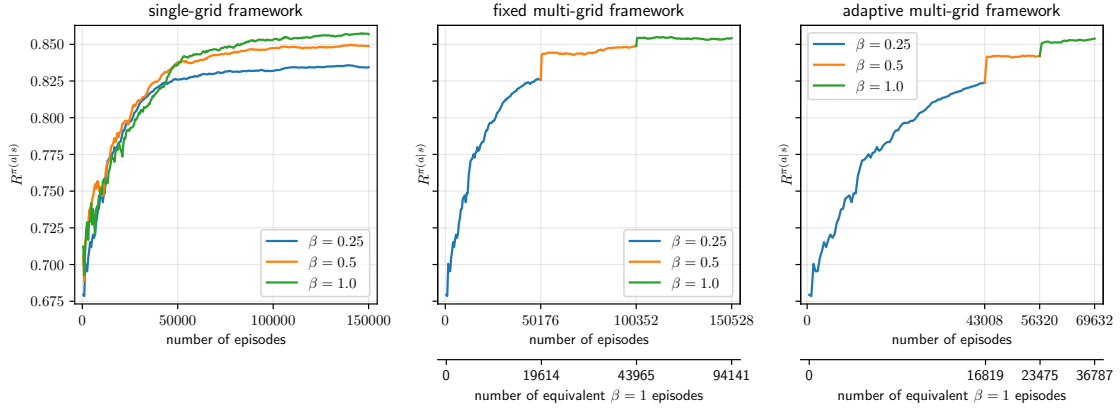


(b)

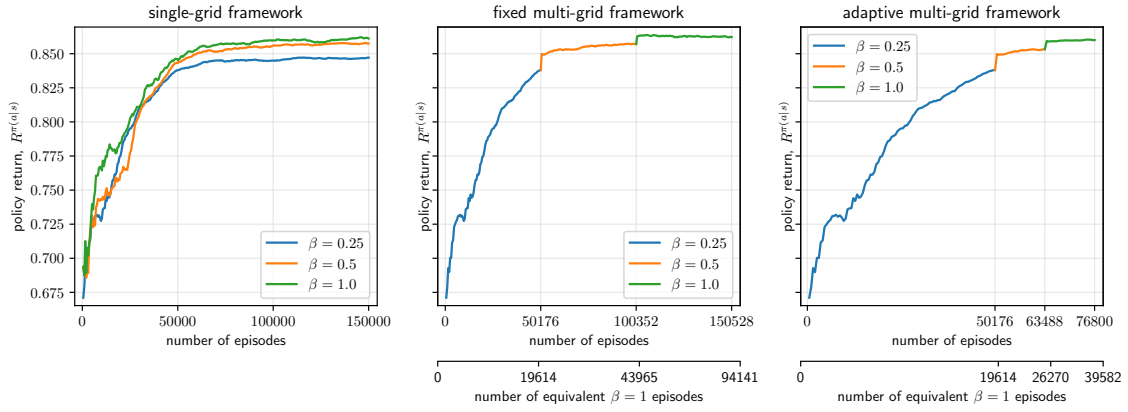


(c)

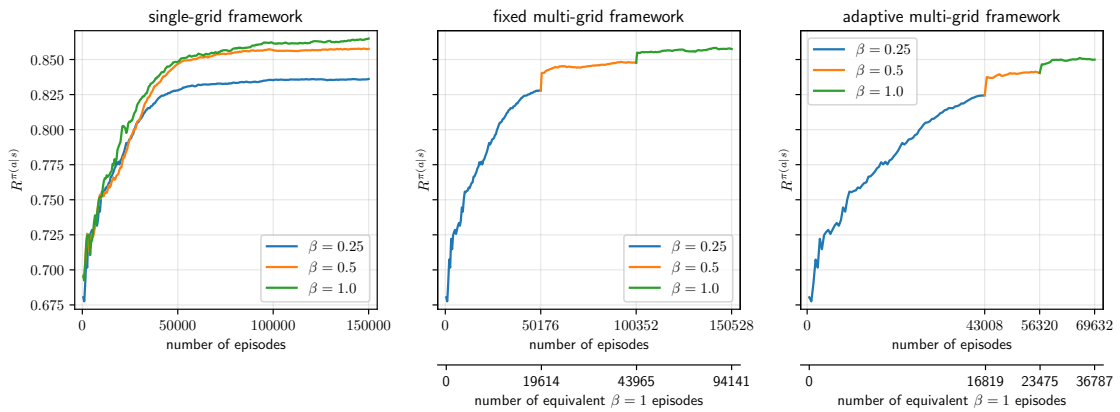
Figure B.2: Learning plots with seed 1 (a), 2 (b) and 3 (c) for test case 1



(a)



(b)



(c)

Figure B.3: Learning plots with seed 1 (a), 2 (b) and 3 (c) for test case 2

Appendix C

Appendices for Chapter 4

C.1 Examples of objective functions for different deep RL algorithms

Examples of the objective function $\mathbb{E}_{s,a,r \sim p_\theta} [J(s, a, r; \theta, \Theta)]$ for various deep reinforcement learning algorithms are delineated in table C.1. In a value-based algorithm, such as the deep Q network (DQN), the neural network represents a function approximator for the Q function. Q function represents the expected return when the agent takes action a_t in state s_t and is defined as $Q(s, a) = \mathbb{E}_\pi [\sum_m \gamma^m r_{m+t+1} | s_t = s, a_t = a]$ where $\gamma \in [0, 1]$ is the discount factor and $\mathbb{E}_\pi[\dots]$ denotes the expected value given that the agent follows the policy π . The policy refers to taking the action corresponding to the highest Q value. In policy based algorithms like advantage actor-critic (A2C), trust region policy optimization (TRPO) and proximal policy optimization (PPO). The policy is directly modeled as a neural network that maps the state s to the corresponding optimal action a . This network is often integrated with a value network, which maps the state s to its corresponding value $V(s)$. The value function is the expected future return for a particular state s_t and is defined as $V(s) = \mathbb{E}_\pi [\sum_m \gamma^m r_{m+t+1} | s_t = s]$. The policy network objective function corresponds to advantage weighted log-likelihood of chosen actions, where advantage function is defined as the difference between Q-function and value function. Algorithms such as TRPO and PPO employ importance sampling to correct for the estimation of the advantage function according to the old policy $\pi_{\theta_{old}}$ (that is, the policy before it is updated in a given policy iteration). As a result, the policy objective function contains the ratio term $\mathbf{r}(\theta) = \pi_\theta(a|s)/\pi_{\theta_{old}}(a|s)$. Subsequently, the objective function for the integrated network is the sum of policy objective function added and value loss term multiplied by value coefficient c_v . In the TRPO algorithm, the destructive steps of large gradients often encountered in policy gradient algorithms such as A2C are avoided by penalizing the KL-divergence between old and new policies with the factor β . In the PPO algorithm, this is achieved by clipping the ratio $\mathbf{r}(\theta)$ between $1 - \epsilon$ and $1 + \epsilon$ for a small value of $\epsilon \in [0, 1]$. Furthermore, the exploration in policy search is maximized by maximizing the entropy of the learned policy

$S[\pi_\theta](s)$ and is added in the objective function with the entropy coefficient c_e .

Table C.1: Objective function $\mathbb{E}_{s,a,r \sim p_\theta}[J(s, a, r; \theta, \Theta)]$ for different deep RL algorithms

| Algorithm | Objective function, $\mathbb{E}_{s,a,r \sim p_\theta}[J(s, a, r; \theta, \Theta)]$ |
|----------------------------------|---|
| DQN (value network) | $\mathbb{E}_{s,a,r \sim p_\theta} [(r + \gamma \max_{a'} Q_{\theta_{old}}(s', a') - Q_\theta(s, a))^2]$ |
| A2C (policy + value network) | $\mathbb{E}_{s,a,r \sim p_\theta} [\log \pi_\theta(a s)A(s, a) - c_v (r + \gamma \max_{s'} V_{\theta_{old}}(s') - V_\theta(s))^2]$ |
| TRPO (policy + value network) | $\mathbb{E}_{s,a,r \sim p_\theta} [\mathbf{r}(\theta)A(s, a) - \beta \text{KL}[\pi_{\theta_{old}}(\cdot a), \pi_\theta(\cdot a)] - c_v (r + \gamma \max_{s'} V_{\theta_{old}}(s') - V_\theta(s))^2]$ |
| PPO (policy + value network) | $\mathbb{E}_{s,a,r \sim p_\theta} [\min(\mathbf{r}(\theta)A(s, a), \text{clip}(\mathbf{r}(\theta), 1 - \epsilon, 1 + \epsilon)A(s, a)) - c_v (r + \gamma \max_{s'} V_{\theta_{old}}(s') - V_\theta(s))^2 + c_e S[\pi_\theta](s)]$ |

C.2 Principle behind computational savings of MLMC estimator

Suppose that we estimate the expectation of the quantity $P^L(\omega)$ where ω is a random variable that follows the probability distribution Ω (that is, $\omega \sim \Omega$). The Monte Carlo estimate of this quantity is given by $\widehat{\mathbb{E}}_\Omega^{MC}(P^L(\omega)) = N^{-1} \sum_{i=1}^N P^L(\omega_i)$. If C and V , respectively, correspond to the computational cost and variance of the term $P^L(\omega_i)$, the cost of the estimator $\widehat{\mathbb{E}}_\Omega^{MC}(P^L(\omega))$ is CN while its overall variance is VN^{-1} . That is, to achieve an overall variance of ϵ^2 , we need to choose $N = \epsilon^{-2}V$ (that is, $N \propto V$). Now, if we suppose that we have an approximation of $P^L(\omega)$ defined as $P^l(\omega)$ such that $\mathbb{V}_\Omega[P^l(\omega)] \gg \mathbb{V}_\Omega[P^L(\omega) - P^l(\omega)]$, the two-level Monte Carlo estimator can be written as $\widehat{\mathbb{E}}_\Omega^{2LMC}(P^L(\omega)) = N_l^{-1} \sum_{i=1}^{N_l} P^l(\omega_i) + N_L^{-1} \sum_{i=1}^{N_L} P^L(\omega_i) - P^l(\omega_i)$. If C_L and V_L are the computational cost and variance of the term $P^L(\omega_i) - P^l(\omega_i)$ while C_l and V_l are the computational cost and variance of the term $P^l(\omega_i)$. The total cost of this two-level Monte Carlo estimator can be computed as $N_l C_l + N_L C_L$ where $N_l \propto V_l$ and $N_L \propto V_L$. Since, by definition, $V_l \gg V_L$, we can also conclude that $N_l \gg N_L$. In other words, if $C_l \ll C_L$, computational cost of two-level Monte Carlo estimate $\widehat{\mathbb{E}}_\Omega^{2LMC}(P^L(\omega))$ is much smaller than the Monte Carlo estimate $\widehat{\mathbb{E}}_\Omega^{MC}(P^L(\omega))$. The same concept can be extended to multilevel Monte Carlo instead of two-level Monte Carlo estimate.

C.3 Implementation of multilevel PPO in stable baselines 3

The multilevel PPO algorithm is implemented using the Stable Baselines3 (SB3) [66] library, which is a set of reliable implementations of reinforcement learning algorithms in PyTorch. The codes for the multilevel implementation can be found in the fork: <https://github.com/atishdixit16/stable-baselines3>. In the following text, the implementation of the classical PPO in SB3 is explained in detail. Then it is followed by additional implementations corresponding to the multilevel PPO algorithm.

C.3.1 Classical PPO implementation in stable baselines 3

RL framework consists of the environment \mathcal{E} which is governed by a Markov decision process described by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \mu \rangle$. Here, $\mathcal{S} \subset \mathbb{R}^{n_s}$ is the state-space, $\mathcal{A} \subset \mathbb{R}^{n_a}$ is the action-space, $\mathcal{P}(s'|s, a)$ is a Markov transition probability function between the current state s and the next state s' under action a and $\mathcal{R}(s, a, s')$ is the reward function. The function $\mu(s)$ returns a state from the initial state distribution if s is the terminal state of the episode; otherwise, it returns the same state s . The goal of reinforcement learning is to find the policy $\pi_\theta(a|s)$ to take an optimal action a when in the state s , by exploring the state-action space with what are called agent-environment interactions. Figure C.1 shows a typical schematic of such agent-environment interaction. The term *agent* refers to the controller that follows the policy $\pi_\theta(a|s)$ while the *environment* consists of the transition function, \mathcal{P} , and the reward function, \mathcal{R} .

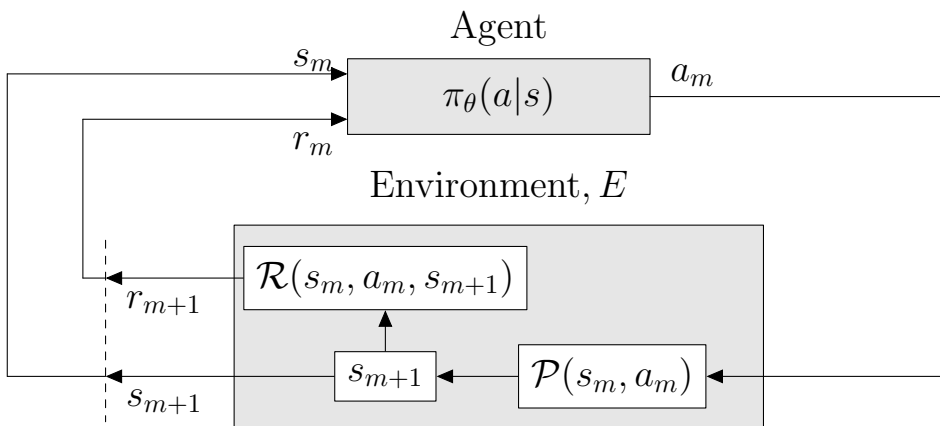


Figure C.1: A typical agent-environment interaction for classical framework

The algorithm 9 delimits the simplified implementation of the PPO algorithm in SB3. The algorithm's inputs are: environment E , number of actors N , number of steps in each policy iteration T , batch size M ($\leq NT$) and number of epochs K . The data obtained through the rollouts of agent-environment interactions is stored in a buffer named RolloutBuffer in the format $[s, a, r, d, V, L_{\text{old}}, R, A]$, where the notation is

- s : state,
- a : action,
- r : reward,
- d : episode terminal boolean (done),
- V : Value function (obtained from policy network rollout),
- L_{old} : log probability value, $\log(\pi_{\theta_{\text{old}}}(a|s))$
- R : Return value (obtained using generalized advantage estimation),
- A : Advantage function (obtained using generalized advantage estimation).

RolloutBuffer accumulates in total $N \times T$ rows of the above data in each iteration. At the beginning of each iteration, the function **CollectRollouts** is used to fill in the data in RolloutBuffer. The total of $N \times T$ data rows is divided into batches of size M , each using the function **GetBatches**. The actor loss term L_a , the value loss term L_v and the entropy loss term L_e (defined in equation 4.5) are calculated for each such batch using the function **ComputeBatchLosses**. Finally, a Monte Carlo estimate for the loss term is computed as follows.

$$\text{loss}_{\text{MC}} = \text{mean}[L_a + L_v + L_e],$$

which is used to update the policy parameters using automatic differentiation. This is done using the function **UpdatePolicy** and is performed K times for every batch.

Algorithm 9 PPO implementation in stable baselines

```

1: Input:  $E, N, T, M, K$ 
2:  $E.reset()$ 
3: Generate empty RolloutBuffer
4: for iteration,  $i = 1, 2, \dots$  do
5:   CollectRollouts( $E, N, T, \text{RolloutBuffer}$ )
6:   for  $epoch = 1, 2, \dots, K$  do
7:     for batch in GetBatches(RolloutBufferArray,  $M$ ): do
8:        $L_a, L_v, L_e = \text{ComputeBatchLosses}$ (batch)
9:        $\text{loss}_{\text{MC}} = \text{mean}[L_a + L_v + L_e]$ 
10:      UpdatePolicy(  $\text{loss}_{\text{MC}}$  )
11:    end for
12:  end for
13: end for

```

The algorithm 10 delineates the steps of the function **CollectRollouts**. For every timestep, the data is obtained using policy rollout, environment transition (using *step* function) and generalized advantage estimation (GAE) computation on all N actors and stored in the RolloutBuffer. Finally, **ComputeBatchLosses** function is illustrated in the algorithm 11. The algorithm lists steps to compute actor loss term L_a , value loss term L_v and entropy loss term L_e for the given batch. Note that the loss terms are the vectors of dimension M , which are added later, and its

Algorithm 10 CollectRollouts($E, N, T, \text{RolloutBuffer}$)

-
- 1: Information: a RolloutBuffer consists of following data: $[s, a, r, d, V, L_{\text{old}}, R, A]$
 - 2: reset RolloutBuffer (i.e. empty the buffer)
 - 3: **for** t in range(T): **do**
 - 4: rollout current state s , through policy network to obtain $a, V, L_{\text{old}}(a)$ on N actors
 - 5: if s is terminal, $s = E.\text{reset}()$
 - 6: $s', r, d, \cdot = E.\text{step}(a)$ on N actors
 - 7: compute R and A using GAE
 - 8: add $[s, a, r, d, V, L_{\text{old}}, R, A]$ in the RolloutBuffer
 - 9: **end for**
-

mean is treated as the final loss term. The mean function in this process indicates the *Monte Carlo* estimator of the PPO loss term.

Algorithm 11 ComputeBatchLosses(batch)

-
- 1: Information: a batch consists of M rows following data: $[s, a, V, L_{\text{old}}, R, A]$
 - 2: compute V_{now} and $L_{\text{now}}(a)$ by rolling out s through policy network
 - 3: compute ratio, $r_t = \exp(L_{\text{now}} - L_{\text{old}})$
 - 4: compute $L_1 = Ar_t$ and $L_2 = A[\text{clip}(r_t, 1 - \epsilon, 1 + \epsilon)]$
 - 5: $L_a = \min(L_1, L_2)$
 - 6: $L_v = C_v |V_{\text{now}} - R|^2$ (C_v is value loss term coefficient)
 - 7: $L_e = -C_e L_{\text{now}}$ (C_e is entropy loss term coefficient)
 - 8: **return** L_a, L_v, L_e
-

The class inheritance schema used in this implementation is shown in Figure C.2. The stable baselines use some more classes like Policy, Callbacks etc. but we present only the ones relevant to this discussion. **CollectRollouts** function belongs to OnPolicyAlgorithm which is the child of the BaseAlgorithm class and the parent of the PPO class. The functions **ComputeBatchLosses** and **UpdatePolicy** belong to the PPO class. BaseBuffer is the parent class for the RolloutBuffer class that contains the function **GetBatches**. The Environment class (which is a child of the gym.Env class) contains functions such as *step* and *reset* corresponding to the transition function \mathcal{P} and the initial state function μ , respectively.

C.3.2 Multilevel PPO implementation in stable baselines 3

Figure C.3 illustrates a typical agent-environment interaction in multilevel PPO implementation. Multiple levels of environment are represented with E^1, E^2, \dots, E^{L-1} so that the computational cost of \mathcal{P}^l and the accuracy of \mathcal{R}^l are lower than \mathcal{P}^{l+1} and \mathcal{R}^{l+1} , respectively. The environment corresponding to the grid fidelity factor l consists of a transition function \mathcal{P}_l , which is achieved by discretizing the dynamical system, and a reward function \mathcal{R}_l . The policy network is designed with states s^L and controls a^L , corresponding to the environment E^L . As a result, state s_{m+1}^l , in the environment, E^l passes through the mapping ψ_l^L which maps the state from level l to level L . Similarly, the action obtained from the policy network is passed through a mapping operator ϕ_L^l , which maps the action from the level L to the level l .

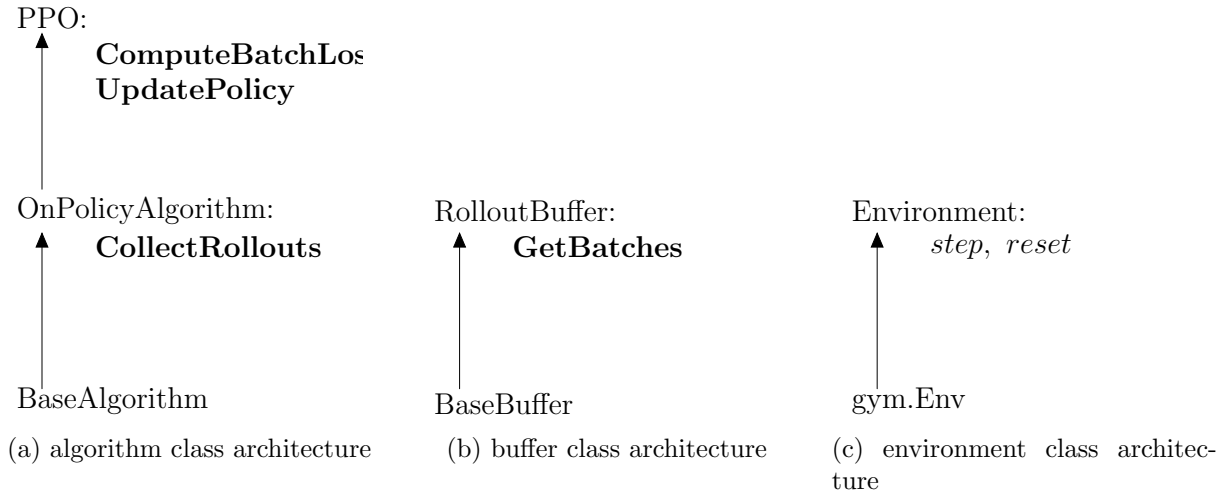


Figure C.2: Object-oriented design for the stable baselines implementation of PPO algorithm

Algorithm 12 illustrates the pseudocode for multilevel implementation of the PPO algorithm in the stable baselines library. The inputs are the same as in classical PPO implementation except multilevel variables are provided as an array of length L : environments at each level $\mathbf{E} = [E^1, E^2, \dots, E^L]$, number of actors N , number of steps in each level $\mathbf{T} = [T^1, T^2, \dots, T^L]$, number of batches in each level $\mathbf{M} = [M^1, M^2, \dots, M^L]$ (such that $NT^l \leq M^l$ and $T^1/M^1 = \dots = T^L/M^L$) and number of epochs K . In multilevel implementation, we formulate the loss term's estimate using multilevel Monte Carlo which is given as

$$\text{loss}_{\text{MLMC}} = \sum_{l=1}^L \text{mean} \left[(L_a^l - \tilde{L}_a^{l-1}) + (L_v^l - \tilde{L}_v^{l-1}) + (L_e^l - \tilde{L}_e^{l-1}) \right],$$

where $\tilde{L}_a^0, \tilde{L}_v^0$ and \tilde{L}_e^0 are set to zero. The outline of a typical agent-environment interaction to obtain synchronized samples of levels l and $l-1$ is illustrated in Figure C.3. We use arrays of RolloutBuffers for each level, and each RolloutBuffer l that collects rollouts at level l has a synchronized buffer SyncRolloutBuffer l that collects corresponding synchronized data at level $l-1$. This is achieved using the function **CollectRollouts**. Figure C.4 illustrates the RolloutBufferArray and SyncRolloutBufferArray used in this algorithm. Furthermore, the **GetBatches** function is used to generate an array of batches, which is used to compute the multilevel Monte Carlo estimate of the loss term. The batch array consists of in total NT^L/M^L batches, where each batch consists of L batches from RolloutBuffers and L batches from SyncRolloutBuffers. Figure C.5 illustrates the batch array used in the algorithm. The batch l , syncBatch $^{l-1}$ from RolloutBuffer l , SyncRolloutBuffer l are used to compute the $\text{loss}_{\text{MLMC}}$ terms on the level l . In every batch, these terms are computed at each level and added to obtain $\text{loss}_{\text{MLMC}}$, which is used to update the policy network parameters using the function **UpdatePolicy**.

The algorithm 13 delimits the function **CollectRollouts** used in multilevel implementation. At each level l the RolloutBuffer l is filled with the data, and the corresponding synchronized data at the level $l-1$ is filled in the SyncRolloutBuffer l . Since L_a^0, L_v^0 and L_e^0 are set to

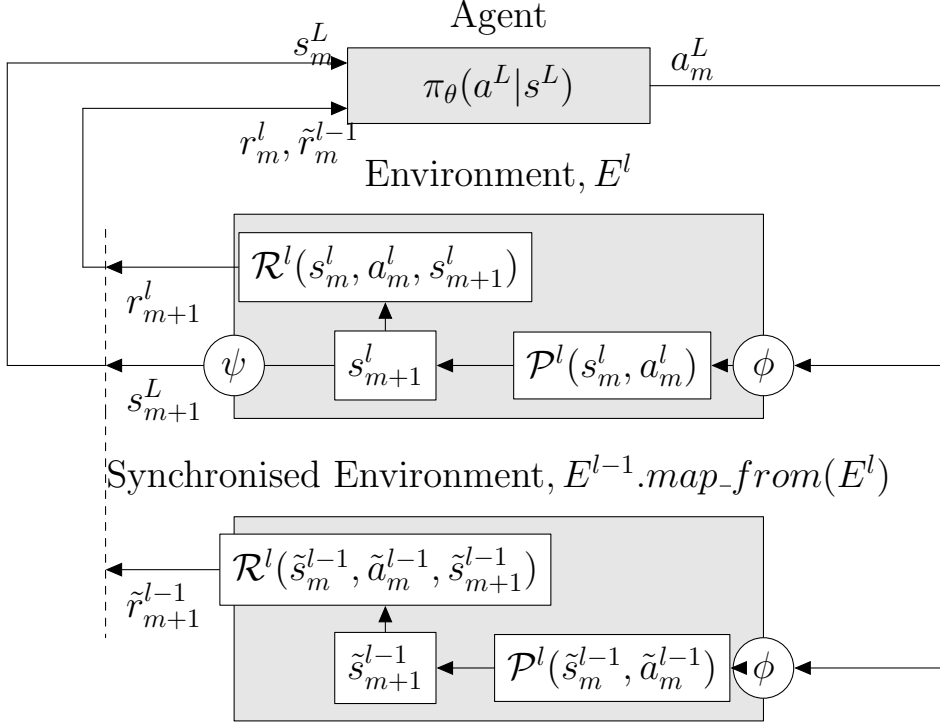


Figure C.3: A typical agent-environment interaction for an environment on level l synchronized with environment on level $l - 1$

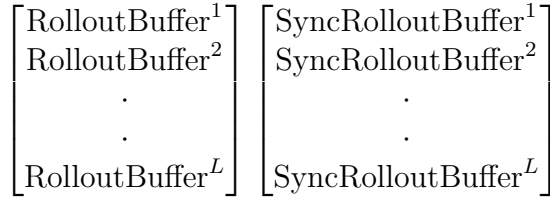


Figure C.4: RolloutBufferArray (on left) and SyncRolloutBufferArray (on right). SyncRolloutBuffer ^{l} consists of synchronized data of RolloutBuffer ^{l} with level l to a level $l - 1$. Each buffer with level l consists of $N \times T_l$ rows of data in $[s, a, r, d, V, L_{\text{old}}, R, A]$ format.

zero, the data in SyncRolloutBuffer¹ are filled with None values. The mapping functions ψ_l^l and ϕ_l^l are implemented as a set of functions in the definition of the environment E^l . As a result, the mapping of state (ψ_l^l from Equation 4.4) and action (ϕ_l^l from equation 4.4) to and from the policy + value network is denoted with shorthand notation ψ and ϕ , respectively. Synchronization of state from level l to l' is indicated by *map_from* function that maps an environment E^l to another environment at level l' , denoted as $E^{l'}$. Algorithm 14 illustrates the pseudocode for the **GetBatches** function, which creates mini-batches (as illustrated in Figure C.5) from collected data in RolloutBufferArray and SyncRolloutBufferArray.

The class inheritance schema used in the multilevel implementation is shown in figure C.6. **CollectRollouts** function belongs to OnPolicyAlgorithmMultilevel which is the child of BaseAlgorithm class and the parent of the PPO_ML class. The functions **ComputeBatchLosses** and **UpdatePolicy** belong to the class PPO_ML. BaseBuffer is the parent class for the Roll-

$$\left[\begin{array}{c} \left[\begin{array}{c} \text{batch}^1, \text{syncBatch}^0 \\ \text{batch}^2, \text{syncBatch}^1 \\ \vdots \\ \text{batch}^L, \text{syncBatch}^{L-1} \end{array} \right] \\ \left[\begin{array}{c} \text{batch}^1, \text{syncBatch}^0 \\ \text{batch}^2, \text{syncBatch}^1 \\ \vdots \\ \text{batch}^L, \text{syncBatch}^{L-1} \end{array} \right] \\ \left[\begin{array}{c} \dots \\ \dots \\ \dots \\ \dots \end{array} \right] \\ \left[\begin{array}{c} \text{batch}^1, \text{syncBatch}^0 \\ \text{batch}^2, \text{syncBatch}^1 \\ \vdots \\ \text{batch}^L, \text{syncBatch}^{L-1} \end{array} \right] \end{array} \right]$$

Figure C.5: `batch_array` which is achieved from `GetBatches` function. It consists of in total NT_l/M_l batches as shown with the columns of the array. Each such batch consists of L batches from RolloutBuffers (denoted by batch^l) and SyncRolloutBuffers (denoted by syncBatch^{l-1}). batch^l and SyncBatch^{l-1} consists of M^l rows of data in the format, $[o, a, V, L_{\text{old}}, R, A]$.

Algorithm 12 Multilevel proximal policy optimization pseudocode

```

1: Input:  $\mathbf{E}, N, \mathbf{T}, \mathbf{M}, K$ 
2:  $E^1.reset()$ 
3: Generate empty RolloutBufferArray, SyncRolloutBufferArray
4: for iteration,  $i = 1, 2, \dots$  do
5:   CollectRollouts( $\mathbf{E}, N, \mathbf{T}$ , RolloutBufferArray, SyncRolloutBufferArray)
6:   for  $epoch = 1, 2, \dots, K$  do
7:     for batch_array in GetBatches(RolloutBufferArray, SyncRolloutBufferArray,  $\mathbf{M}$ ):
8:       do
9:          $\text{loss}_{\text{MLMC}} = 0$ 
10:        for  $\text{batch}^l, \text{syncBatch}^{l-1}$  in batch_array do
11:           $L_a^l, L_v^l, L_e^l = \text{ComputeBatchLosses}(\text{batch}^l)$ 
12:          if  $l > 1$  then
13:             $\tilde{L}_a^{l-1}, \tilde{L}_v^{l-1}, \tilde{L}_e^{l-1} = \text{ComputeBatchLosses}(\text{syncBatch}^{l-1})$ 
14:          else
15:             $\tilde{L}_a^{l-1}, \tilde{L}_v^{l-1}, \tilde{L}_e^{l-1} = 0$ 
16:          end if
17:           $L^l = \text{mean} \left[ (L_a^l - \tilde{L}_a^{l-1}) + (L_v^l - \tilde{L}_v^{l-1}) + (L_e^l - \tilde{L}_e^{l-1}) \right]$ 
18:           $\text{loss}_{\text{MLMC}} = \text{loss}_{\text{MLMC}} + L^l$ 
19:        end for
20:        UpdatePolicy(  $\text{loss}_{\text{MLMC}}$  )
21:      end for
22:    end for

```

Algorithm 13 CollectRollouts($\mathbf{E}, N, \mathbf{T}$, RolloutBufferArray, SyncRolloutBufferArray)

```

1: Information: a RolloutBuffer consists of following data:  $[s, a, r, d, V, L_{\text{old}}, R, A]$ 
2: reset RolloutBufferArray, SyncRolloutBufferArray (i.e. empty the buffers)
3: for  $T^l, E^l, \text{RolloutBuffer}^l, \text{SyncRolloutBuffer}^l$  in  $\mathbf{E}, \mathbf{T}$ , RolloutBufferArray, SyncRollout-
   BufferArray do
4:   if  $l > 1$  then
5:      $E^l.map\_from(E^{l-1})$ 
6:   end if
7:   for  $t$  in range( $T^l$ ): do
8:      $s^l = E^l.reset()$  if  $s^l$  is terminal
9:      $s^L = E^l.\psi(s^l)$ 
10:     $a^L = \pi_\theta(a^L|s^L)$ 
11:     $a^l = \phi(a^L)$ 
12:    compute  $V^l$  and  $L_{\text{old}}(a^L)$ 
13:     $\cdot, r^l, d^l, \cdot = E^l.step(a^l)$  on  $N$  actors
14:    compute  $R^l$  and  $A^l$  using GAE
15:    add  $[s^l, a^l, r^l, d^l, V^l, L_{\text{old}}^l, R^l, A^l]$  in the RolloutBuffer $^l$ 
16:
17:    if  $l > 1$  then
18:       $E^{l-1}.map\_from(E^l)$ 
19:       $\tilde{s}^L = E^{l-1}.\psi(\tilde{s}^{l-1})$ 
20:       $\tilde{a}^{l-1} = a^l$ 
21:       $\tilde{a}^L = \pi_\theta(\tilde{a}^L|\tilde{s}^L)$ 
22:       $\tilde{a}^{l-1} = E^{l-1}.\phi(\tilde{a}^L)$ 
23:      compute  $\tilde{V}^{l-1}$  and  $\tilde{L}_{\text{old}}(a^L)$ 
24:       $\cdot, \tilde{r}^{l-1}, \cdot, \cdot = E^{l-1}.step(\tilde{a}^{l-1})$  on  $N$  actors
25:      compute  $\tilde{R}^{l-1}$  and  $\tilde{A}^{l-1}$  using GAE
26:      add  $[\tilde{s}^{l-1}, \tilde{a}^{l-1}, \tilde{r}^{l-1}, \tilde{d}^l, \tilde{V}^{l-1}, \tilde{L}_{\text{old}}^{l-1}, \tilde{R}^{l-1}, \tilde{A}^{l-1}]$  in the SyncRolloutBuffer $^l$ 
27:    else
28:      add [None, ..., None] in the SyncRolloutBuffer $^l$ 
29:    end if
30:   end for
31: end for

```

Algorithm 14 GetBatches(RolloutBufferArray, SyncRolloutBufferArray, \mathbf{M})

```

1: set batch_array to an empty array
2: for RolloutBuffer $^l, \text{SyncRolloutBuffer}^l, M^l$  in RolloutBufferArray, SyncRolloutBufferArray,
    $\mathbf{M}$  do
3:   set batches to an empty array
4:   for batch $^l, \text{batch}^{l-1}$  in GetSyncBatches(RolloutBuffer $^l, \text{SyncRolloutBuffer}^l, M^l$ ) do
5:     batches.append([batch $^l, \text{batch}^{l-1}$ ])
6:   end for
7:   batch_array.append(batches)
8: end for
9: return batch_array

```

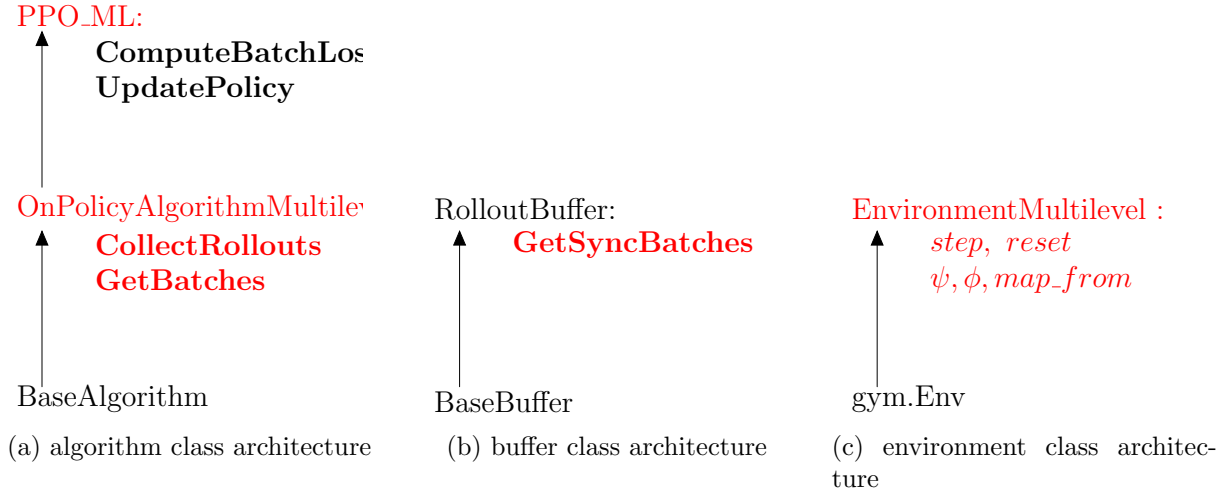


Figure C.6: Object-oriented design for the stable baselines implementation of multilevel PPO algorithm. The updated (from classical PPO implementation) definitions of functions and classes are highlighted in red colour.

outBuffer class that contains the function `GetBatches`. The environment class architecture for multilevel framework is similar to that for classical framework except for the additional mapping functions ψ , ϕ and `map_from`. The updated definitions of the classes and functions are highlighted in red in figure C.6.

C.4 Cluster analysis of permeability uncertainty distribution

A set of permeability samples $\mathbf{k} = \{k_1, \dots, k_l\}$, is chosen to represent the variability in the permeability distribution \mathcal{K} . For the optimal control problem, our main interest is the uncertainty in the dynamical response of permeability, rather than the uncertainty in permeability itself. As a result, the connectivity distance [84] is used as a measure of the distance between the permeability field samples. The connectivity distance matrix $\mathbf{D} \in \mathbb{R}^{N \times N}$ among the N samples of \mathcal{K} is formulated as

$$\mathbf{D}(k_i, k_j) = \sum_{x''} \int_{t_0}^T [c(x'', t; k_i) - c(x'', t; k_j)]^2 dt,$$

where N corresponds to a large number of samples of uncertainty distribution, $c(x'', t; k_i)$ is the concentration at the location x'' and at time t , when the permeability is set to k_i and all wells are open equally. The multidimensional scaling of the distance matrix \mathbf{D} is used to produce N two-dimensional coordinates d_1, d_2, \dots, d_N , each representing a permeability sample. The coordinates d_1, d_2, \dots, d_N are obtained such that the distance between d_i and d_j is equivalent to $\mathbf{D}(k_i, k_j)$. In the k-means clustering process, these coordinates are divided into

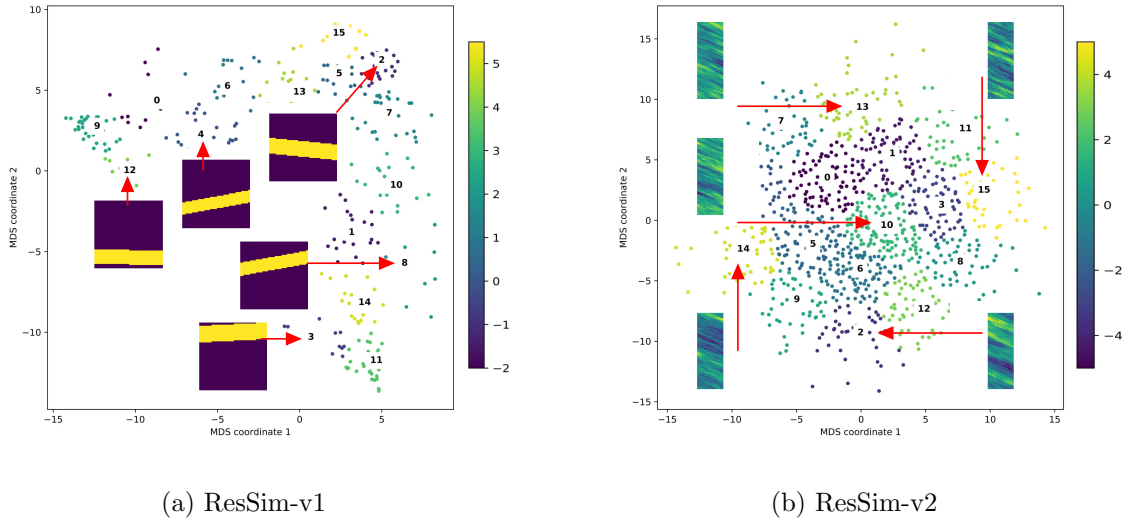


Figure C.7: clustering visualization for permeability samples

l sets S_1, S_2, \dots, S_l , obtained by solving the optimization problem:

$$\arg \min_S \sum_i^l \sum_{d_j \in S_i} \|d_j - \mu_{S_i}\|,$$

where μ_{S_i} is the average of all coordinates in the set S_i . The training vector \mathbf{k} is a set of l samples of \mathcal{K} where each of its values k_i corresponds to the closest one to μ_{S_i} . The total number of samples N and clusters l is chosen to be 1000 and 16 for both uncertainty distributions, \mathcal{G}_1 and \mathcal{G}_2 . A training vector \mathbf{k} is obtained with samples k_1, \dots, k_{16} each corresponding to a cluster center. Figures C.7a and C.7b show cluster plots of permeability samples for ResSim-v1 and ResSim-v2.

C.5 Algorithm Parameters

Parameters used for PPO are tabulated in Table C.2 which were tuned using trial and error. For PPO algorithms, the parameters were essentially tuned to find the least variability in the learning plots. The parameters of the DE algorithm are delineated in Table C.3. The code repository for both test cases presented in this article can be found at the link: https://github.com/atishdixit16/multilevel_ppo.

Table C.2: PPO algorithm parameters

| | ResSim-v1 | ResSim-v2 |
|-------------------------------------|--------------------|------------------|
| discount rate, γ | 0.99 | 0.99 |
| clip range, ϵ | 0.1 | 0.15 |
| policy network MLP layers | [93,150,100,80,62] | [35,70,70,50,21] |
| policy network activation functions | tanh | tanh |
| policy network optimizers | Adam | Adam |
| learning rate | 3e-6 | 1e-5 |

Table C.3: DE algorithm parameters

| | ResSim-v1 | ResSim-v2 |
|----------------------|-----------|-----------|
| number of CPUs | 64 | 64 |
| number of iterations | 1024 | 1024 |
| population size | 310 | 105 |
| recombination factor | 0.9 | 0.9 |
| mutation factor | (0.5,1) | (0.5,1) |

Bibliography

- [1] David Silver et al. “Mastering chess and shogi by self-play with a general reinforcement learning algorithm”. In: *arXiv preprint arXiv:1712.01815* (2017).
- [2] Hyeong Soo Chang et al. “Google Deep Mind’s AlphaGo”. In: *OR/MS Today* 43.5 (2016), pp. 24–29.
- [3] Johannes Dornheim, Norbert Link, and Peter Gumbsch. “Model-free adaptive optimal control of episodic fixed-horizon manufacturing processes using reinforcement learning”. In: *International Journal of Control, Automation and Systems* 18.6 (2020), pp. 1593–1604.
- [4] Enrico Anderlini et al. “Control of a point absorber using reinforcement learning”. In: *IEEE Transactions on Sustainable Energy* 7.4 (2016), pp. 1681–1690.
- [5] Jean Rabault et al. “Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control”. In: *Journal of fluid mechanics* 865 (2019), pp. 281–302.
- [6] Hans O Jahns. “A rapid method for obtaining a two-dimensional reservoir description from well pressure response data”. In: *Society of Petroleum Engineers Journal* 6.04 (1966), pp. 315–327.
- [7] ML Wasserman and AS Emanuel. “History matching three-dimensional models using optical control theory”. In: *Journal of Canadian Petroleum Technology* 15.04 (1976).
- [8] ROBERT J Watson et al. “Rhizobium meliloti genes required for C4-dicarboxylate transport and symbiotic nitrogen fixation are located on a megaplasmid”. In: *Journal of Bacteriology* 170.2 (1988), pp. 927–934.
- [9] Dean S Oliver and Yan Chen. “Recent progress on reservoir history matching: a review”. In: *Computational Geosciences* 15.1 (2011), pp. 185–221.
- [10] AS Lee and JS Aronofsky. “A linear programming model for scheduling crude oil production”. In: *Journal of Petroleum Technology* 10.07 (1958), pp. 51–54.
- [11] JS Aronofsky and AC Williams. “The use of linear programming and mathematical models in under-ground oil production”. In: *Management Science* 8.4 (1962), pp. 394–407.
- [12] Robert A Wattenbarger and HJ Ramey. “An investigation of wellbore storage and skin effect in unsteady liquid flow: II. Finite difference treatment”. In: *Society of Petroleum Engineers Journal* 10.03 (1970), pp. 291–297.

- [13] Zohreh Fathi and W Fred Ramirez. “Optimal injection policies for enhanced oil recovery: Part 2—surfactant flooding”. In: *Society of Petroleum Engineers Journal* 24.03 (1984), pp. 333–341.
- [14] Sebastián Eloy Sequeira, Moisès Graells, and Luis Puigjaner. “Real-time evolution for on-line optimization of continuous processes”. In: *Industrial & engineering chemistry research* 41.7 (2002), pp. 1815–1825.
- [15] Edgar Chacón, Isabel Besembel, and Jean Claude Hennet. “Coordination and optimization in oil and gas production complexes”. In: *Computers in Industry* 53.1 (2004), pp. 17–37.
- [16] Vidar Gunnerud and Bjarne Foss. “Oil production optimization—A piecewise linear model, solved with two decomposition strategies”. In: *Computers & Chemical Engineering* 34.11 (2010), pp. 1803–1812.
- [17] Mohammad Sadegh Tavallali et al. “Optimal producer well placement and production planning in an oil reservoir”. In: *Computers & chemical engineering* 55 (2013), pp. 109–125.
- [18] DR Brouwer et al. “Improved reservoir management through optimal control and continuous model updating”. In: *SPE annual technical conference and exhibition*. OnePetro. 2004.
- [19] Louis J Durlofsky and Khalid Aziz. “Optimization of smart well control”. In: *SPE international thermal operations and heavy oil symposium and international horizontal well technology conference*. OnePetro. 2002.
- [20] JFBM Kraaijevanger et al. “Optimal waterflood design using the adjoint method”. In: *SPE Reservoir Simulation Symposium*. OnePetro. 2007.
- [21] Diego F Oliveira and Albert Reynolds. “An adaptive hierarchical multiscale algorithm for estimation of optimal well controls”. In: *SPE Journal* 19.05 (2014), pp. 909–930.
- [22] YX Wang et al. “Optimization in oilfield water injection system based on algorithm of ant colony-particle swarm method”. In: *J. Daqing Pet. Inst* 2 (2010), p. 014.
- [23] Daoyong Yang, Qi Zhang, and Yongan Gu. “Integrated optimization and control of the production-injection operation systems for hydrocarbon reservoirs”. In: *Journal of petroleum science and Engineering* 37.1-2 (2003), pp. 69–81.
- [24] GM Van Essen et al. “Robust waterflooding optimization of multiple geological scenarios”. In: *Spe Journal* 14.01 (2009), pp. 202–210.
- [25] Morteza Haghghat Sefat et al. “Reservoir uncertainty tolerant, proactive control of intelligent wells”. In: *Computational Geosciences* 20.3 (2016), pp. 655–676.
- [26] Bahare Kiumarsi et al. “Optimal and autonomous control using reinforcement learning: A survey”. In: *IEEE transactions on neural networks and learning systems* 29.6 (2017), pp. 2042–2062.
- [27] Moritz Diehl, Hans Georg Bock, and Johannes P Schlöder. “A real-time iteration scheme for nonlinear optimization in optimal feedback control”. In: *SIAM Journal on control and optimization* 43.5 (2005), pp. 1714–1736.

- [28] Lorenz T Biegler and James Blake Rawlings. *Optimization approaches to nonlinear model predictive control*. Tech. rep. Argonne National Lab., IL (USA), 1991.
- [29] Basil Kouvaritakis and Mark Cannon. *Non-linear Predictive Control: theory and practice*. 61. Iet, 2001.
- [30] Ivan Koryakovskiy et al. “Benchmarking model-free and model-based optimal control”. In: *Robotics and Autonomous Systems* 92 (2017), pp. 81–90.
- [31] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. 2016, pp. 1928–1937.
- [32] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).
- [33] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).
- [34] Anusha Nagabandi et al. “Neural network dynamics for model-based deep reinforcement learning with model-free fine-tuning”. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 7559–7566.
- [35] Thomas Anthony, Zheng Tian, and David Barber. “Thinking fast and slow with deep learning and tree search”. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5360–5370.
- [36] David Ha and Jürgen Schmidhuber. “World models”. In: *arXiv preprint arXiv:1803.10122* (2018).
- [37] Nathan Lambert et al. “Objective mismatch in model-based reinforcement learning”. In: *arXiv preprint arXiv:2002.04523* (2020).
- [38] Volodymyr Mnih et al. “Asynchronous methods for deep reinforcement learning”. In: *International conference on machine learning*. PMLR. 2016, pp. 1928–1937.
- [39] John Schulman et al. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).
- [40] Dan Huang. *How much did AlphaGo-Zero cost?* 2018.
- [41] Alexander Y Sun. “Optimal carbon storage reservoir management through deep reinforcement learning”. In: *Applied Energy* 278 (2020), p. 115660.
- [42] Michael B Giles. “Multilevel monte carlo methods”. In: *Acta numerica* 24 (2015), pp. 259–328.
- [43] Catarina Roseta-Palma and Anastasios Xepapadeas. “Robust control in water management”. In: *Journal of Risk and Uncertainty* 29.1 (2004), pp. 21–34.
- [44] DR Brouwer et al. “Recovery increase through water flooding with smart well technology”. In: *SPE European Formation Damage Conference*. Society of Petroleum Engineers. 2001.
- [45] Stephen Whitaker. “Single-phase flow in homogeneous porous media: Darcy’s Law”. In: *The method of volume averaging*. Springer, 1999, pp. 161–180.
- [46] Fabio Muratore et al. “Robot learning from randomized simulations: A review”. In: *arXiv preprint arXiv:2111.00956* (2021).

- [47] Rainer Storn and Kenneth Price. “Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces”. In: *Journal of global optimization* 11.4 (1997), pp. 341–359.
- [48] John Schulman et al. “High-dimensional continuous control using generalized advantage estimation”. In: *arXiv preprint arXiv:1506.02438* (2015).
- [49] Jef Caers, Kwangwon Park, and Céline Scheidt. “Modeling uncertainty in metric space”. In: *International association of mathematical geology meeting, Stanford University*. 2009.
- [50] Jørg E Aarnes, Tore Gimse, and Knut-Andreas Lie. “An introduction to the numerics of flow in porous media using Matlab”. In: *Geometric modelling, numerical simulation, and optimization*. Springer, 2007, pp. 265–306.
- [51] Michael Andrew Christie, MJ Blunt, et al. “Tenth SPE comparative solution project: A comparison of upscaling techniques”. In: *SPE reservoir simulation symposium*. Society of Petroleum Engineers. 2001.
- [52] C Anderson and S Crawford-Hines. “Multigrid Q-learning”. In: *Technical Report CS-94-121*. Citeseer, 1994.
- [53] Omer Ziv and Nahum Shimkin. “Multigrid methods for policy evaluation and reinforcement learning”. In: *Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation Intelligent Control, 2005*. IEEE. 2005, pp. 1391–1396.
- [54] Stephan Pareigis. “Multi-grid methods for reinforcement learning in controlled diffusion processes”. In: *NIPS*. Citeseer. 1996, pp. 1033–1039.
- [55] Bocheng Li and Li Xia. “A multi-grid reinforcement learning method for energy conservation and comfort of HVAC in buildings”. In: *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2015, pp. 444–449.
- [56] Matthew E Taylor and Peter Stone. “Transfer learning for reinforcement learning domains: A survey.” In: *Journal of Machine Learning Research* 10.7 (2009).
- [57] Sanmit Narvekar et al. “Source task creation for curriculum learning”. In: *Proceedings of the 2016 international conference on autonomous agents & multiagent systems*. 2016, pp. 566–574.
- [58] Alessandro Lazaric, Marcello Restelli, and Andrea Bonarini. “Transfer of samples in batch reinforcement learning”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 544–551.
- [59] Fernando Fernández, Javier García, and Manuela Veloso. “Probabilistic policy reuse for inter-task transfer learning”. In: *Robotics and Autonomous Systems* 58.7 (2010), pp. 866–871.
- [60] Anestis Fachantidis et al. “Transferring task models in reinforcement learning agents”. In: *Neurocomputing* 107 (2013), pp. 23–32.
- [61] Matthew E Taylor and Peter Stone. “Behavior transfer for value-function-based reinforcement learning”. In: *Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*. 2005, pp. 53–59.

- [62] Ruslan Miftakhov, Abdulaziz Al-Qasim, and Igor Efremov. “Deep reinforcement learning: reservoir optimization from pixels”. In: *International Petroleum Technology Conference*. OnePetro. 2020.
- [63] Yusuf Nasir et al. “Deep reinforcement learning for constrained field development optimization in subsurface two-phase flow”. In: *arXiv preprint arXiv:2104.00527* (2021).
- [64] Atish Dixit and Ahmed H ElSheikh. “Stochastic optimal well control in subsurface reservoirs using reinforcement learning”. In: *Engineering Applications of Artificial Intelligence* 114 (2022), p. 105106.
- [65] Sebastian Müller and Lennart Schüler. *GeoStat-Framework/GSTools: Bouncy Blue*. Version v1.0.0. Jan. 2019. DOI: [10.5281/zenodo.2541735](https://doi.org/10.5281/zenodo.2541735). URL: <https://doi.org/10.5281/zenodo.2541735>.
- [66] Antonin Raffin et al. *Stable Baselines3*. <https://github.com/DLR-RM/stable-baselines3>. 2019.
- [67] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: [10.1038/s41592-019-0686-2](https://doi.org/10.1038/s41592-019-0686-2).
- [68] Atish Dixit and Ahmed H. ElSheikh. “Stochastic optimal well control in subsurface reservoirs using reinforcement learning”. In: *Engineering Applications of Artificial Intelligence* 114 (2022), p. 105106. ISSN: 0952-1976. DOI: <https://doi.org/10.1016/j.engappai.2022.105106>. URL: <https://www.sciencedirect.com/science/article/pii/S0952197622002469>.
- [69] K Andrew Cliffe et al. “Multilevel Monte Carlo methods and applications to elliptic PDEs with random coefficients”. In: *Computing and Visualization in Science* 14.1 (2011), pp. 3–15.
- [70] David F Anderson and Desmond J Higham. “Multilevel Monte Carlo for continuous time Markov chains, with applications in biochemical kinetics”. In: *Multiscale Modeling and Simulation* 10.1 (2012), pp. 146–179.
- [71] Michael B Giles and Lukasz Szpruch. “Multilevel Monte Carlo methods for applications in finance”. In: *High-Performance Computing in Finance* (2018), pp. 197–247.
- [72] Yuyang Shi and Rob Cornish. “On multilevel Monte Carlo unbiased gradient estimation for deep latent variable models”. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2021, pp. 3925–3933.
- [73] Neil K Chada et al. “Multilevel Bayesian Deep Neural Networks”. In: *arXiv preprint arXiv:2203.12961* (2022).
- [74] Matthijs TJ Spaan. “Partially observable Markov decision processes”. In: *Reinforcement Learning*. Springer, 2012, pp. 387–414.
- [75] Farzad Hourfar et al. “A reinforcement learning approach for waterflooding optimization in petroleum reservoirs”. In: *Engineering Applications of Artificial Intelligence* 77 (2019), pp. 98–116.

- [76] JL Guevara, Rajan G Patel, and Japan J Trivedi. “Optimization of steam injection for heavy oil reservoirs using reinforcement learning”. In: *SPE International Heavy Oil Conference and Exhibition*. OnePetro. 2018.
- [77] Jin-Hee Lee and John W Labadie. “Stochastic optimization of multireservoir systems via reinforcement learning”. In: *Water resources research* 43.11 (2007).
- [78] Jincong He et al. “Deep reinforcement learning for generalizable field development optimization”. In: *SPE Journal* 27.01 (2022), pp. 226–245.
- [79] Kai Zhang et al. “Training effective deep reinforcement learning agents for real-time life-cycle production optimization”. In: *Journal of Petroleum Science and Engineering* 208 (2022), p. 109766.
- [80] Yusuf Nasir and Louis J Durlofsky. “Deep reinforcement learning for optimal well control in subsurface systems with uncertain geology”. In: *arXiv preprint arXiv:2203.13375* (2022).
- [81] Atish Dixit and Ahmed H Elsheikh. “Robust Optimal Well Control using an Adaptive Multigrid Reinforcement Learning Framework”. In: *Mathematical Geosciences* (2022), pp. 1–31.
- [82] Atish Dixit and Ahmed Elsheikh. “A multilevel reinforcement learning framework for PDE based control”. In: *arXiv preprint arXiv:2210.08400* (2022).
- [83] Jian Hou et al. “A review of closed-loop reservoir management”. In: *Petroleum Science* 12.1 (2015), pp. 114–128.
- [84] Kwangwon Park. *Modeling uncertainty in metric space*. Stanford University, 2011.
- [85] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. “Numba: A llvm-based python jit compiler”. In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. 2015, pp. 1–6.