NEW TECHNIQUES FOR REDUCED-ORDER MODELING IN RESERVOIR SIMULATION

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF ENERGY RESOURCES ENGINEERING AND THE COMMITTEE ON GRADUATE STUDIES OF STANFORD UNIVERSITY IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

Zhaoyang Larry Jin
August 2019
I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Louis Durlofsky, Primary Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Daniel Tartakovsky

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

Oleg Volkov

Approved for the Stanford University Committee on Graduate Studies.

Patricia J. Gumport, Vice Provost for Graduate Education

This signature page was generated electronically upon submission of this dissertation in electronic format. An original signed hard copy of the signature page is on file in University Archives.
Abstract

Reservoir simulation is widely applied for the management of oil and gas production and CO\textsubscript{2} storage operations. It can be computationally expensive, however, particularly when the flow physics is complicated and many simulation runs must be performed. This has motivated the development of reduced-order modeling (ROM) procedures, where the goal is to achieve high degrees of computational speedup along with reasonable solution accuracy. In this work we develop and apply two types of ROM methods – one based on proper orthogonal decomposition (POD) and piecewise linearization, and one based on deep learning.

We first develop a POD-based ROM, referred to as POD-TPWL, to simulate coupled flow-geomechanics problems. In POD-TPWL, proper orthogonal decomposition, which enables the representation of solution unknowns in a low-dimensional subspace, is combined with trajectory piecewise linearization (TPWL), where solutions with new sets of well controls are represented via linearization around previously simulated (training) solutions. The over-determined system of equations is projected into the low-dimensional subspace using a least-squares Petrov-Galerkin procedure. The states and derivative matrices required by POD-TPWL, generated by an extended version of Stanford’s Automatic-Differentiation-based General Purpose Research Simulator, are provided in an offline (pre-processing or training) step. Offline computational requirements correspond to the equivalent of 5-8 full-order simulations, depending on the number of training runs used. Runtime (online) speedups of $O(100)$ or more are achieved for new POD-TPWL test-case simulations. The POD-TPWL model is tested extensively for a 2D coupled problem involving oil-water flow and geomechanics. It is shown that POD-TPWL provides predictions of reasonable accuracy, relative to full-order simulations, for well-rate quantities, global pressure
and saturation fields, global maximum and minimum principal stress fields, and the Mohr-Coulomb rock failure criterion, for the cases considered. A systematic study of POD-TPWL error is conducted using various training procedures for different levels of perturbation between test and training cases. The use of randomness in the well bottom-hole pressure profiles used in training is shown to be beneficial in terms of POD-TPWL solution accuracy. The procedure is also successfully applied to a prototype 3D example case.

We next apply the POD-TPWL reduced-order modeling framework to simulate and optimize the injection stage of CO₂ storage operations. The use of multiple derivatives, meaning that the linearizations are performed around different training solutions at different time steps, is described and assessed. Two example cases are presented, and the ability of the POD-TPWL model to accurately capture bottom-hole pressure, when time-varying CO₂ injection rates are prescribed, is demonstrated. It is also shown that, for these examples, the reduced-order models can provide accurate estimates of CO₂ molar fraction at particular locations in the domain. The POD-TPWL model is then incorporated into a mesh adaptive direct search optimization framework where the objective is to minimize the amount of CO₂ reaching a target layer at the end of the injection period. The POD-TPWL model is shown to be well suited for this purpose and to provide optimization results that are comparable to those obtained using full-order simulations. POD-TPWL preprocessing computations entail a (serial) time equivalent of about 6.7 full-order simulations, though the resulting runtime speedups, relative to full-order simulation, are about 100–150 for the cases considered.

Finally, we develop a new deep-learning-based ROM for reservoir simulation. The reduced-order model is based on an existing embed-to-control (E2C) framework and includes an auto-encoder, which projects the system to a low-dimensional subspace, and a linear transition model, which approximates the evolution of the system states in low dimension. In addition to the loss function for data mismatch considered in the original E2C framework, we introduce a physics-based loss function that penalizes predictions that are inconsistent with the governing flow equations. The loss function is also modified to emphasize accuracy in key well quantities of interest (e.g., fluid
production rates). The E2C ROM is shown to have interesting parallels with POD-TPWL. The new ROM is applied to oil-water flow in a 2D heterogeneous reservoir. A total of 300 high-fidelity training simulations are performed in the offline stage, and the network training requires 10-12 minutes on a Tesla V100 GPU node. Online (runtime) computations achieve speedups of $O(1000)$ relative to full-order simulations. Extensive test case results, with well controls varied over large ranges, are presented. Accurate ROM predictions are achieved for global saturation and pressure fields at particular times, and for injection and production well responses as a function of time. Error is shown to increase when 100 or 200 (rather than 300) training runs are used to construct the E2C ROM.
Acknowledgments

First, I want to express my sincere gratitude to my advisor, Prof. Louis J. Durlofsky, who offered his inspiration, encouragement and guidance throughout my six years at Stanford. His rigorous research approach, spirit of dedication and work ethic have truly inspired me over the years. His advice and support have helped me go through the most struggling moments in my research. I appreciate the fact that he tolerated those non-productive days and provided the freedom for me to explore new ideas. Especially, I want to thank him for the significant amount of patience he offered while working with me. It has been an honor and pleasure to work with him.

I want to thank the reading members of my defense committee, Prof. Daniel Tartakovsky and Dr. Oleg Volkov, for the effort spent reviewing this thesis. I also want to thank Prof. Khalid Aziz for serving as the non-reading member of my defense committee and Prof. Gregory Beroza for kindly chairing my oral defense. I really appreciate their service.

Furthermore, my gratitude goes to all my collaborators and those who offered help in my research projects. I want to thank Dr. Jincong He, a graduated Ph.D. student in our research group, who provided a tremendous amount of help on my research, and answered all my questions with respect to the research project, even the most naive ones, with patience. I want to thank Dr. Oleg Volkov, Dr. Timur Garipov and Yimin Liu, who are the co-authors of my papers. It has been truly rewarding and inspiring to work with them. I want to thank Dr. Yuke Zhu, Dr. Nicola Castelletto, Dr. Vladislav Bukshtynov, Meng Tang, Dr. Dave Cameron, Dr. Sumeet Trehan, Dr. Forest (Rui) Jiang, Dr. Wenyue Sun, Dr. Francois Pascal Hamon and Dr. Ogy Grujic for their help in my research. Their insight and knowledge lent me a strong hand in solving various types of problems.
I want to thank my colleagues and friends at the Stanford ERE department. My thanks go to my current and former officemates, Dr. Usua Amanam, Hannah Lu, Kelly (Mengyu) Guan, Dr. Michael Connolly, Dr. Zhi-Yang Wang, Dr. Kyu Koh Yoo, Dr. Muhammad Almajid, and Ala Alzayer, with whom I spent plenty of quality time and had a lot of fun. My thanks go to all the ERE current and former students, staff members and faculty for creating such a friendly and inspiring academic environment. I want to thank Prof. Warren Kourt, Prof. Kiran Pande, Dr. Amir Delgoshaie, Dr. Chuan Tian and Haoli Guo, particularly for sincere advice and help for my career development.

I want to thank my supporting groups during my study at Stanford, including the Chinese community within ERE and School of Earth Science (SCAPE), and the Chinese Entrepreneurship Organization (CEO). I am truly grateful to all my friends at Stanford. I am also grateful to all my friends outside Stanford, especially my book club members for those deep discussions and reflections on all aspects of life. I would like to acknowledge the financial support provided by Stanford Smart Fields Consortium (SFC) and SUPRI-B, which made all my research possible.

I want to thank all my colleagues and people I worked with during my internships. Especially, I want to thank Dr. Dominic Camilleri (Halliburton), Dr. Bradley Mallison (Chevron), Dr. Pierre Muron (Chevron), Min Na (Wood Mackenzie), Dr. Tingyun Yang (Wood Mackenzie), Lin Wan (EIF), Yue Li (EIF), Tommy Woolley (IOG Capital), Charlie Shufeldt (IOG Capital), and Dr. Jingbo Wang (IDG Capital) for their guidance and help.

Finally, I want to thank my father Shuigen Jin, and my mother Qing Huang, who offered unconditional love and support. Most important of all, I would like to thank my beloved girlfriend, Dr. Sherry Juexiao Wang, who is always there for me whenever I need her.
# Contents

Abstract v

Acknowledgments viii

1 Introduction 1
   1.1 Literature Review ........................................ 2
   1.1.1 POD-based reduced-order models ......................... 2
   1.1.2 Carbon sequestration ................................ 6
   1.1.3 Coupled flow-geomechanics ............................. 9
   1.1.4 Deep-learning-based ROM ................................ 10
   1.2 Scope of Work ............................................. 13
   1.3 Dissertation Outline .................................... 14

2 POD-TPWL for Coupled Flow-Geomechanics 16
   2.1 POD-TPWL Formulation ................................. 17
   2.1.1 Governing equations .................................... 17
   2.1.2 Linearized representation ............................. 21
   2.1.3 POD and constraint reduction ......................... 22
   2.1.4 Detailed numerical treatments ....................... 25
   2.1.5 POD-TPWL workflow .................................. 29
   2.2 POD-TPWL Simulation Results ........................... 29
   2.2.1 Model 2.1: 2D example ............................... 30
   2.3 Impact of Training Runs on POD-TPWL Accuracy ............ 42
   2.4 Model 2.2: 3D Example .................................... 51
   2.5 Summary ................................................. 58
5 Conclusions and Future Work 134
  5.1 Summary and Conclusions ........................................... 134
  5.2 Recommendations for Future Work ............................... 137

Appendix A Embed-to-Control Network Architecture 140

Appendix B Additional Test Case Results 143
  B.1 Results for Test Case 2 .......................................... 143
  B.2 Results for Test Case 3 .......................................... 144

Bibliography 169
List of Tables

3.1 POD-TPWL BHP errors (in bar) for five test cases using varying numbers of derivatives (Model 3.1) ........................................ 78
3.2 POD-TPWL $z_{CO_2}$ errors for five test cases using varying numbers of derivatives (Model 3.1) ........................................ 79
3.3 POD-TPWL errors for five test cases (Model 3.2) ..................... 86
4.1 Errors and percentiles for test cases (Model 4.1) ........................ 133
A.1 Network architecture for encoder ........................................... 141
A.2 Network architecture for decoder ........................................... 141
List of Figures

2.1 2D example (Model 2.1) setup ........................................... 31
2.2 Injection schedules for training and test runs (Model 2.1) .......... 32
2.3 Well rates for test case (Model 2.1) .................................... 34
2.4 Pressure at 400 days for Model 2.1 (unit: bar) ...................... 36
2.5 Oil saturation at 400 days (Model 2.1) ................................. 37
2.6 Maximum effective principal stress ($\sigma'_1$) at 400 days for Model 2.1 (unit: bar) ................................................................. 38
2.7 Minimum effective principal stress ($\sigma'_3$) at 400 days for Model 2.1 (unit: bar) ................................................................. 39
2.8 Mohr-Coulomb failure criterion ($\chi$) at 400 days (Model 2.1) .... 41
2.9 Mohr-Coulomb failure criterion ($\chi$) at the injection wells (Model 2.1) .......................................................... 43
2.10 Variation of selected training run with time (Model 2.1) ............ 43
2.11 Training schedules for ‘3 regular trainings’ procedure ($\gamma_{ptb} = 20$ bar), along with test-case schedule ................................. 45
2.12 Alternative training schedules for Well I2 ($\gamma_{ptb} = 20$ bar) ........ 45
2.13 Mohr-Coulomb failure criterion ($\chi$) for injection wells ($\gamma_{ptb} = 20$ bar). For ‘3 regular trainings’ $E_{\chi} = 0.119$, and for ‘5 random trainings’ $E_{\chi} = 0.067$ ....................................................... 48
2.14 Systematic error assessment for Model 2.1. Legend in (b) applies to all subfigures ................................................................. 50
2.15 3D model (Model 2.2) setup ................................................. 52
2.16 Injection well BHP schedules for training and test runs (Model 2.2) .......................................................... 53
2.17 Well rates for test case (Model 2.2) ........................................ 54
2.18 Oil saturation in layer 2 at 600 days (Model 2.2) ...................... 55
2.19 Maximum effective principal stress ($\sigma'_1$) in layer 2 at 600 days (Model 2.2) 56
2.20 Minimum effective principal stress ($\sigma'_3$) in layer 2 at 600 days (Model 2.2) 57
2.21 Mohr-Coulomb failure criterion ($\chi$) at the injection wells (Model 2.2) 58

3.1 POD-TPWL with derivatives from two training runs 65
3.2 Relative permeability curves used in all simulations 69
3.3 Model 3.1 setup (Gaussian log-permeability field) 70
3.4 Injection schedules at reservoir conditions for training runs (Model 3.1) 72
3.5 Injection schedules for test cases (Model 3.1) 73
3.6 BHP responses for Test Case 1, Wells 1 and 2 (Model 3.1) 74
3.7 BHP responses for Test Case 1, Wells 3 and 4 (Model 3.1) 75
3.8 Overall molar fraction of CO$_2$ at the end of injection in layer 6 for Test Case 1 (Model 3.1) 76
3.9 Aggregate CO$_2$ in layer 6 for Test Case 1 (Model 3.1) 77
3.10 Model 3.2 setup (channelized permeability field) 79
3.11 Injection schedules at reservoir conditions for training runs (Model 3.2) 81
3.12 Injection schedules for test cases (Model 3.2) 82
3.13 BHP responses for Test Case 6, Wells 1 and 2 (Model 3.2) 83
3.14 BHP responses for Test Case 6, Wells 3 and 4 (Model 3.2) 84
3.15 Overall molar fraction of CO$_2$ at the end of injection in layer 3 for Test Case 6 (Model 3.2) 85
3.16 Aggregate CO$_2$ in layer 3 for Test Case 6 (Model 3.2) 86
3.17 Results for minimization of aggregate CO$_2$ (Model 3.2) 89
3.18 Initial guess and optimized well controls (Model 3.2) 90
3.19 CO$_2$ molar fraction in layer 3 at the end of injection (20 years, Model 3.2) 90
3.20 Cross-sectional views of CO$_2$ distribution at the end of injection (20 years, Model 3.2) 92
3.21 Differences in initial guess and optimal solutions (left), and POD-TPWL model error (right), for Wells 1 and 3 (Model 3.2) 93

4.1 Embed-to-control (E2C) overview 102
4.2 Encoder layout 104
4.3 Linear transition model layout 107
4.4 Decoder layout .......................................................... 111
4.5 Pressure field predictions with and without $\mathcal{L}_p$ (all colorbars in units
of psi) ............................................................................. 113
4.6 Model 4.1: log-permeability field and well locations ............... 117
4.7 Test Case 1: well controls (Model 4.1) ............................... 120
4.8 Test Case 1: saturation field at 200 days (Model 4.1) .......... 121
4.9 Test Case 1: saturation field at 1000 days (Model 4.1) ......... 122
4.10 Test Case 1: saturation field at 1800 days (Model 4.1) ....... 124
4.11 Test Case 1: pressure field at 1000 days (all colorbars in units of psi,
Model 4.1) .......................................................................... 125
4.12 Test Case 1: production rates for Well P1 ......................... 126
4.13 Test Case 1: production rates for Well P2 (Model 4.1) ........ 126
4.14 Test Case 1: injection BHPs (Model 4.1) ......................... 127
4.15 Cumulative oil and water production for all 100 test cases (Model 4.1) 129
4.16 Errors for quantities for interest (Model 4.1) ....................... 131
4.17 ROM error with different numbers of training runs (Model 4.1) 132
A.1 Blocks in embed-to-control models ................................. 142
B.1 Test Case 2: saturation field at 200 days (Model 4.1) ........... 144
B.2 Test Case 2: saturation field at 1000 days (Model 4.1) ......... 145
B.3 Test Case 2: saturation field at 1800 days (Model 4.1) ........ 146
B.4 Test Case 2: pressure field at 1000 days (all colorbars in units of psi,
Model 4.1) .......................................................................... 147
B.5 Test Case 2: production rates for Well P1 (Model 4.1) ........... 148
B.6 Test Case 2: production rates for Well P2 (Model 4.1) .......... 148
B.7 Test Case 2: injection BHPs (Model 4.1) ......................... 149
B.8 Test Case 3: saturation field at 200 days (Model 4.1) ........... 150
B.9 Test Case 3: saturation field at 1000 days (Model 4.1) ......... 151
B.10 Test Case 3: saturation field at 1800 days (Model 4.1) ........ 152
B.11 Test Case 3: pressure field at 1000 days (all colorbars in units of psi,
Model 4.1) ......................................................................... 153
B.12 Test Case 3: production rates for Well P1 (Model 4.1) ........... 154
B.13 Test Case 3: production rates for Well P2 (Model 4.1) . . . . . . . . . . . . . 154
B.14 Test Case 3: injection BHPs (Model 4.1) . . . . . . . . . . . . . . . . . . . . 155
Reservoir simulation is a key technology for reservoir performance prediction and management, and it has been widely applied in oil and gas production, carbon sequestration, and coupled flow-geomechanics problems. Reservoir simulation entails numerically solving a set of governing partial differential equations (PDEs) that describe flow in the subsurface. However, these computations can be quite time-consuming due to the nonlinear nature of the governing PDEs. This is especially true in simulations of CO$_2$ storage operations, where compositional models with complicated phase behavior are often applied, and in coupled flow-geomechanics problems, where sets of PDEs representing multiple physical processes are solved in a coupled manner. In addition, realistic models may contain $O(10^5 - 10^7)$ or more grid cells, so the number of equations to be solved is very large when there are multiple unknowns in the problem. The computational burden is particularly demanding in applications such as optimization and uncertainty quantification, where hundreds or thousands of simulation runs may be required.

A variety of surrogate models have been developed to address the aforementioned issues. Surrogate models refer to proxy modeling techniques that can approximate the solution of a full-order (high-fidelity) model with significant speedup. In the context of reservoir simulation, surrogate models that have been developed include response surface methods [122], upscaled or multiscale methods [24, 26], and fast marching methods [120], among others. Reduced-order modeling (ROM) methods refer to a
CHAPTER 1. INTRODUCTION

type of surrogate model that achieves computational speedup by reducing the number of solution variables in the system. These methods first require training, during which some number of full-order runs are performed to construct the surrogate representation. Deep neural networks [75] have also been used to improve the performance of a wide range of surrogate models.

In this dissertation, we first focus on the development of a POD-based reduced-order modeling method called POD-TPWL. In POD-TPWL, proper orthogonal decomposition (POD), which projects the unknown variables to a low-dimensional subspace, is combined with trajectory piecewise linearization (TPWL), in which nonlinear effects are represented via local linearization around previously simulated (training) solutions. We develop POD-TPWL for CO$_2$ storage simulations and then apply it to provide significant speedup in an optimization procedure. POD-TPWL is also applied to coupled flow-geomechanics problems to predict key quantities of interest. Finally, we introduce an embed-to-control (E2C) ROM, which is a deep-learning-based procedure, and apply it to oil-water problems. This method requires several hundred training simulation runs, though it is then able to provide accurate predictions for cases that are very different from the training runs.

1.1 Literature Review

We now discuss both POD-based and deep-learning-based ROM techniques. POD-based ROMs and their application in optimization problems are first discussed. Efficient simulation and optimization for carbon sequestration is then reviewed. Next, we discuss ROMs for coupled flow-geomechanics problems. Finally, deep-learning techniques and their use for reservoir simulation problems are considered.

1.1.1 POD-based reduced-order models

Reduced-order modeling procedures represent a broad family of computational techniques that have received significant attention in recent years. The discussion here will focus on a popular category of ROMs, which are based on proper orthogonal decomposition (POD). These ROM methods, referred to as POD-based ROMs, typically involve an offline (preprocessing) component, where training runs are performed and
relevant solution information is processed and saved, and an online (runtime) component, where new (test) runs are performed. These POD-based methods typically entail three key procedures: solution representation in low dimension, constraint reduction, and nonlinearity treatment. POD is commonly applied to enable the low-dimensional representation of solution unknowns in the online computations. A method for the projection of the system of equations, also referred to as constraint reduction, is additionally required. The two widely used treatments for this are Galerkin projection and least-squares Petrov-Galerkin (LSPG) projection.

So-called POD-only methods include the use of POD to represent solution unknowns, plus a constraint reduction treatment. POD-only was first applied in linear subsurface flow problems by Vermeulen et al. [109], and in oil-water (nonlinear) problems by Heijn et al. [51] and van Doren et al. [108]. These approaches all used Galerkin projection for constraint reduction. In subsequent work, Cardoso et al. [16] applied a POD-only procedure to a 3D water-flooding problem with 60,000 cells (also with Galerkin projection for constraint reduction). They achieved improved efficiency through use of snapshot clustering and missing point estimation (MPE) procedures, though speedups were still less than a factor of ten. These modest speedups were due to the need to perform expensive matrix multiplications at each Newton iteration in the nonlinear loop. Such computations are required in all POD-only procedures.

Various approaches have been applied to treat solution nonlinearity. The goal of these procedures is to eliminate the expensive (runtime) matrix multiplications required by POD-only methods. One such approach is the discrete empirical interpolation method (DEIM), proposed by Chaturantabut and Sorensen [23]. DEIM reduces the computation associated with POD-only methods by approximating nonlinear functions based on their values at relatively few interpolation points. Gildin et al. [39] first applied POD-DEIM in a reservoir simulation setting. Subsequently, Ghasemi and Gildin [37], Yang et al. [121] and Ghommem et al. [38] extended POD-DEIM to more complicated oil-water systems, where a speedup of up to six fold was reported.

The Gauss-Newton with approximated tensor (GNAT) method, developed by Carlberg et al. [18], represents a generalization of DEIM for treating nonlinearity. DEIM requires the number of columns in the nonlinear basis to be equal to the
number of interpolation points, while GNAT does not. GNAT has been applied for structural and solid mechanics [124], electromechanics [2], and computational fluid dynamics [19]. Yoon et al. [123] first applied GNAT for both 2D and 3D oil-water reservoir simulation problems. Jiang [56] and Jiang and Durlofsky [57] performed detailed parametric studies for GNAT on oil-water reservoir simulations. Tang [105] implemented GNAT within a general purpose reservoir simulation setting and tested the procedure with 2D oil-water and 3D compositional models. Typical speedups in this study were around a factor of 2-3, though this could likely be improved somewhat with a more efficient implementation. Another method for nonlinearity treatment is the radial basis function (RBF) multidimensional interpolation method proposed by Xiao et al. [119]. This approach is referred to as POD-RBF.

Trajectory piecewise linearization, originally introduced by Rewienski and White [94], treats nonlinearity differently than GNAT, POD-DEIM or POD-RBF. TPWL was initially combined with a Krylov projection for solution representation, and was then applied for circuit design problems. Bechtold et al. [8] combined TPWL with POD instead of Krylov projection. This procedure is now referred to as POD-TPWL. Specifically, POD-TPWL entails linearization around ‘nearby’ training solutions (the iterative solution of nonlinear equations is entirely avoided), which allows for very high degrees of speedup. Cardoso and Durlofsky [14] first applied POD-TPWL for subsurface flow simulation (3D oil-water models with up to 79,200 grid blocks). They achieved reasonable solution accuracy and reported speedups of $O(500)$. Subsequently, He et al. [48] improved the accuracy and the stability of POD-TPWL by incorporating specialized treatments, including a local resolution procedure and a so-called equal density projection technique. These methods were tested in 3D oil-water models with up to 108,000 grid blocks. For constraint reduction, Galerkin projection was used by both Cardoso and Durlofsky [14] and He et al. [48]. He and Durlofsky [47] subsequently implemented a least-squares Petrov-Galerkin (LSPG) projection and demonstrated significantly improved simulation stability in comparison to Galerkin projection. POD-TPWL has also been implemented for more complicated subsurface flow simulations involving oil-gas compositional systems [46, 47] and idealized thermal recovery problems [97].

Recent developments related to POD-TPWL have also been reported. Xiao et al.
1.1. LITERATURE REVIEW

[118] integrated POD-TPWL with RBF and domain decomposition and presented a nonintrusive subdomain POD-TPWL method for reservoir history matching. The approach was tested on a 2D prototype oil-water model with 2500 grid blocks and Gaussian-distributed log-permeability fields. POD-TPWL has also been extended to include a quadratic term, which gives a trajectory piecewise quadratic (POD-TPWQ) procedure [107]. Tan et al. [104] developed a trajectory-based DEIM (TDEIM) procedure by combining the advantages of TPWL and DEIM. The procedure was applied to a 2D oil-water problem with 3600 cells.

GNAT and POD-DEIM [123, 121, 27] are intrusive with respect to the simulator, meaning they require access to the full-order simulator at run time. They achieve speedup relative to full-order simulation by (in part) calculating quantities such as Jacobian matrix elements at only a selected subset of grid blocks. POD-TPWL, by contrast, requires extensive output of simulation quantities (states and derivative matrices) during training runs, but it does not require simulator access during online computations. In this sense, it is much less intrusive than POD-DEIM or GNAT procedures. The methods that treat nonlinearity directly (GNAT, POD-DEIM, POD-TPWQ) have been shown to be generally more accurate than POD-TPWL [107, 57], though the highest speedups consistently reported are with POD-TPWL. Thus there exists a tradeoff between solution accuracy and computational efficiency.

Previous work employing ROMs to optimize subsurface flow problems has also been reported. Some of these studies are discussed by Jansen and Durlofsky [54], who reviewed the application of ROMs in well-control optimization. For example, in the work of van Doren et al. [108], a standalone POD method was used for gradient-based well-control optimization in an oil-water problem. Cardoso and Durlofsky [15] and He et al. [48] combined POD-TPWL with a gradient-based algorithm and a generalized pattern search (GPS) approach, respectively, for production optimization in 3D oil-water models. He and Durlofsky [46] applied Hooke-Jeeves direct search with POD-TPWL for production optimization with a 3D compositional model.

In most of the previous POD-TPWL studies mentioned above (including [15, 48, 46]), only one training run (referred to as the primary training run) was used for linearization. POD-TPWL accuracy can however degrade as the controls (and thus test-case solutions) shift further away from the controls used in the primary training
run [107, 57]. Cardoso and Durlofsky [15] and He et al. [48] therefore implemented retraining procedures to improve the accuracy of POD-TPWL during optimization when the difference between the test and training controls became large. Subsequently, Fragoso et al. [33] developed detailed retraining criteria for POD-TPWL-based optimization. Other treatments such as the use of multiple training runs for linearization have also been reported to improve POD-TPWL accuracy [96].

More recently, Pinto et al. [92] applied POD-DEIM for production optimization for a 3D oil-water problem. Trehan and Durlofsky [107] used their trajectory piecewise quadratic method (POD-TPWQ) as an error estimator to determine when retraining was required. They developed a trust-region optimization framework that applied POD-TPWL for function evaluations (for an oil-water optimization problem). Sorek et al. [103] combined POD-DEIM with polynomial approximations, which were used to parameterize and thus reduce the dimension of the optimization variables. This approach was applied to water-flooding optimization problems for a 2D reservoir with about 2600 grid blocks.

1.1.2 Carbon sequestration

One of the application areas for which we will develop and apply a reduced-order modeling procedure is geologic carbon storage. Carbon capture and sequestration (CCS) represents a potential approach for mitigating the impact of anthropogenic CO$_2$. Numerical simulation will be an essential tool in the management of large-scale storage operations, as it can be used to optimize financial and risk measures and to quantify uncertainty in the location and state of the injected CO$_2$. A detailed simulation program will also enable the design of an optimal surveillance strategy, which can in turn be used to reduce key uncertainties and to drive a ‘closed-loop’ aquifer management approach. However, the presence of fine-scale features that can impact flow, along with the uncertainty inherent in the geologic characterization, may lead to excessive demands in terms of computational time. There is thus a need for accelerating the required simulations while retaining sufficient accuracy in results for quantities of interest.

There have been a number of methods presented previously for the efficient simulation of CO$_2$ storage problems. Several such methods are described by Mishra et al.
Many of the existing treatments can be categorized as simplified physical models or statistical approaches. The former include, e.g., the use of Buckley-Leverett-type radial displacement formulations, adapted to model CO$_2$ plume migration [85, 87]. These methods are useful in many settings, but they include simplifying assumptions (radial flow, homogeneous aquifer, fully penetrating vertical well), which render them inappropriate for the simulations considered here. Other types of simplified models, developed by, e.g., Nordbotten and Celia [86] and Gasda et al. [35], are based on a vertical equilibrium assumption and a sharp interface approximation. In more recent work Guo et al. [45] relaxed the vertical equilibrium assumption and introduced a model for vertical dynamics. Although these approaches incorporate effects not included in the simplified radial flow models, they still involve assumptions (e.g., fully penetrating vertical wells) that may limit their application in realistic settings.

Statistical approaches include the work of Wriedt et al. [117], who used a Box-Behnken experimental design procedure to establish a stepwise quadratic regression model for predicting the behavior of injected CO$_2$. In another study, Schuetter et al. [99] used a space-filling max-min Latin hypercube sampling design to predict performance metrics such as total storage efficiency, CO$_2$ plume radius and average reservoir pressure. Both approaches were further developed and validated for CO$_2$ geologic sequestration by Mishra et al. [81] through comparison to the full-physics model. As was the case with the reduced-physics models noted above, these methods again involve simplifying assumptions (e.g., single vertical well in a homogeneous aquifer) that limit their applicability for the problems considered here. Other efficient treatments for CO$_2$ sequestration modeling include the use of a surrogate reservoir model (SRM) with a grid-based approach [84] or with an artificial neural network (ANN) assisted pattern recognition approach [100]. The latter used fully-connected neural networks and was applied to a case study of a large-scale CO$_2$ storage project in the SACROC unit [101].

Optimization of CO$_2$ storage operations has received attention in recent years, and the objective functions considered have involved both risk and economic metrics. For example, Shamshiri and Jafarpour [102] introduced procedures to determine the optimal time-dependent injection rates for multiple CO$_2$ injectors under geological
uncertainty, with the goal of maximizing total stored gas in the aquifer, or maximizing sweep efficiency of the CO\textsubscript{2} flood. Cameron and Durlofsky [11, 12] applied derivative-free optimization methods to determine both the well locations and injection schedules that minimized total CO\textsubscript{2} mobility at the top layer of the storage aquifer. Geological uncertainty and data assimilation were also incorporated in the latter study. Goda and Sato [42] used an iterative Latin hypercube sampling method to optimize the placement of injection wells, with the goal of minimizing the total amount of mobile CO\textsubscript{2} in the aquifer. Pan et al. [88] optimized time-varying injection strategies using a kriging-based meta-modeling technique. Their objective was to maximize the residually trapped volume fraction of CO\textsubscript{2} over multiple realizations. Babaei et al. [5] introduced a procedure to optimize the injection well locations such that the hysteretic trapping of CO\textsubscript{2} was maximized. They employed Monte Carlo simulation and polynomial chaos expansions to handle geological uncertainty.

As noted earlier, typical optimization studies entail large numbers (hundreds to thousands) of function evaluations. If full-order models are used, as in the studies cited above, each function evaluation corresponds to an expensive simulation. Computational requirements can increase as more effects, such as geological uncertainty, are included and more optimization variables are considered. Accurate surrogate models could thus be extremely useful for CO\textsubscript{2} storage optimization. Petvipusit et al. [91] applied adaptive sparse grid interpolation to accelerate the optimization of CO\textsubscript{2} injection rates for an economic objective function, the tax break-even point. Petvipusit et al. [90] subsequently proposed an efficient surrogate-assisted optimization technique with a high-dimensional model representation to maximize the economic value of CO\textsubscript{2} injection in deep saline aquifers. This approach entailed fewer forward simulations to build the surrogate model and also required less computation to determine each update in the optimization procedure. Babaei et al. [6] applied an evolutionary optimization algorithm in conjunction with upscaled models to maximize the sum of dissolved and residually trapped CO\textsubscript{2} in heterogeneous models. Their use of upscaled models reduced the computational intensity of the optimization.

In the author’s Master’s research [58], a prototype POD-TPWL procedure was developed for CO\textsubscript{2} storage operations. The method can treat horizontal wells operating under specified rates (as opposed to specified bottom-hole pressure). The
1.1. LITERATURE REVIEW

POD-TPWL formulation was applied for CO$_2$ injection into a synthetic channelized aquifer and a layered aquifer model. The latter model was constructed based on the Mount Simon formation, which was intended to be used for the FutureGen 2.0 project [9]. The work in [58] involved linearization around a single training run, and did not include any optimization.

1.1.3 Coupled flow-geomechanics

Coupled flow and geomechanics simulations are computationally demanding and thus appropriate for reduced-order modeling. Their computational expense is due to the nonlinear multiphysics nature of the governing equations, and the extended domain over which the geomechanical problem must be solved.

A number of methods for the full-order simulation of coupled flow and geomechanics problems have recently been developed. Kim et al. [66, 67] proposed a sequential implicit solver for this problem. Their approach was further validated and compared to a fully implicit method by Castelletto et al. [22]. Wong et al. [116] extended the concept to a sequential fully implicit Newton solver for multiphysics systems. White et al. [115] presented a preconditioner based on an approximate block-factorization (i.e., block-preconditioned solver) to improve the convergence efficiency of coupled poromechanics problems. This approach was further developed to a block-preconditioned Krylov method by Klevtsov et al. [71], and to a two-stage Newton-Krylov solver by White et al. [114]. More recently, Bui et al. [10] proposed a multigrid reduction framework that algebraically decouples the tightly coupled system with appropriate interpolation and restriction operations. This approach was shown to be efficient for large-scale multiphase poromechanics problems.

A number of methods for the more efficient modeling of geomechanics and coupled flow-geomechanics problems have been presented. Many of these treatments fall into the category of upscaling or multiscale methods. An excellent example is the multiscale finite element method (MSFE) applied to linear elastic geomechanics problems [21]. In that work, a coarse-scale grid is imposed on the fine-scale problem, and coarse-scale basis functions are constructed by solving local fine-scale problems. These basis functions are then used in the solution of the coarse-scale system. Within a coupled flow-geomechanics setting, Castelletto et al. [20] combined MSFE for geomechanics
with the multiscale finite volume method for flow to create a hybrid multiscale finite element-finite volume method. Other types of surrogate modeling methods include semi-analytical proxy models [4] and data-driven models [72].

POD-based reduced-order models have also been developed recently for coupled flow-geomechanics problems. Florez [30] applied LSPG projection with Gauss-Newton iteration (PGGN) to treat thermo-poroelasticity problems. In later work, Florez and Gildin [31] extended the PGGN approach to treat one-way coupled flow-geomechanics (‘one-way’ here means the geomechanical responses are not affected by flow) in single-phase unconventional reservoir models. These ROMs are analogous to POD-only procedures described earlier, in which POD and constraint reduction are applied (but no nonlinear treatment is introduced). As discussed in Section 1.1.1, these methods are limited in that they generally provide very little speedup relative to full-order simulations that utilize state-of-the-art linear solvers. Subsequently, Florez and Gildin [32] applied POD-DEIM to a one-way coupled problem similar to that considered in [31]. They reported reasonable accuracy and improved speedup compared to their earlier results in [31].

1.1.4 Deep-learning-based ROM

The POD-based ROMs described above entail physics-based treatments and numerical approximations. Alternate approaches involve the use of machine learning. In fact, the recent success of deep learning in image processing has inspired the rapid development of algorithms for subsurface modeling that make use of deep neural networks. These methods have been applied for geological parameterization, uncertainty quantification, and surrogate/reduced-order modeling.

For geological parameterization and uncertainty quantification, Canchumuni et al. [13] generated new geological realizations from randomized low-dimensional latent variables using a variational auto-encoder (VAE). A VAE [70] entails a convolutional encoder-decoder neural network architecture, where the encoder component projects a high-dimensional distribution into a low-dimensional random vector, with each element following an independent Gaussian distribution. The decoder acts as the inverse of the encoder and projects the sampled Gaussian-distributed random variables back to the high dimension. Laloy et al. [73] achieved a similar goal using a generative
adversarial network (GAN), where the projection to high dimension is determined by training two adversarial neural networks (known as the generator and the discriminator). Liu et al. [79] and Liu and Durlofsky [78] combined principal component analysis (PCA) based representations with convolutional neural networks (CNN) to give a CNN-PCA procedure. This approach applied the ‘fast neural style transfer’ algorithm [63] to represent complex geological models characterized by multipoint spatial statistics, and was shown to enable more efficient data assimilation.

Zhu and Zabaras [127] formulated surrogate modeling as an image-to-image regression, and constructed a Bayesian deep convolutional neural network for geological uncertainty quantification. Subsequently, Mo et al. [83] extended this model to handle multiphase flow problems. They further improved performance by introducing additional physical constraints.

Recent developments involving the use of deep-learning techniques in ROMs indicate great potential for such approaches. Lee and Carlberg [77] introduced an improved GNAT procedure by replacing POD with AE. The resulting method was applied to a 1D dynamic Burgers’ equation and a 2D quasi-static chemically reacting flow problem, with the boundary conditions in the test runs different from those in the training runs. Kani and Elsheikh [65] developed a deep residual recurrent neural network (DR-RNN) procedure, which employed RNN to approximate the low-dimensional residual functions for the governing equations in a POD-DEIM procedure. The resulting ROM was then applied to a 1D oil-water problem with the distribution of porosity in the test runs perturbed from that of the training runs. Zhang et al. [125] used a fully-connected network to replace the Newton iterations in a POD-DEIM procedure. The method was used to predict well responses in a 2D oil-water problem, in which combinations of well controls and permeability fields for test runs were different from those of the training simulations. Though improvements in accuracy were achieved by all of the above approaches relative to the ‘standard’ implementations, all of these implementations were within existing ROM settings; i.e., none adopted an end-to-end deep-learning framework.

Other researchers have developed ROM methodologies that represent more of a departure from existing approaches. Wang et al. [112], for example, used the
long-short-term-memory (LSTM) RNN [36] to approximate flow dynamics in a low-dimensional subspace constructed by POD. Subsequently, Gonzalez and Balajewicz [43] replaced the POD step with AE for the low-dimensional representation. Both of these approaches, however, were applied on relatively simple problems, where the only differences between online and offline simulation runs were the initial conditions of the systems (boundary conditions were identical). Varying well settings between offline and online computations is an essential feature for ROMs in the context of oil production or CO$_2$ storage optimization, so the above implementations may not be directly applicable in these settings. Another potential limitation is that these procedures are purely data driven and do not take the underlying governing equations into consideration. This could result in solutions that are visually appealing but physically unrealistic.

A number of methods have been applied to incorporate physical constraints into deep neural networks. These procedures have different names but often share the same key ideas. Raissi et al. [93] introduced a physics-informed deep learning framework (later referred to as physics-informed neural network or PINN) that used densely connected feed-forward neural networks. In PINN, the residual functions associated with the governing partial differential equations (PDEs) are introduced into the loss function of the neural network. Zhu et al. [128] extended this PDE-constraint concept to a deep flow-based generative model (GLOW [69]), and constructed a surrogate model for uncertainty quantification using residuals of the governing equations rather than simulation outputs. Watter et al. [113] proposed an embed-to-control (E2C) framework, in the context of robotic planning systems, to predict the evolution of system states using direct sensory data (images). This procedure can handle time-varying controls as inputs. The E2C framework combines a VAE, which is used as both an inference model to project the system states to a low-dimensional subspace, and a generative model to reconstruct the prediction results at full order, with a linear transition model. The latter approximates the evolution of low-dimensional states based on the time-varying control inputs.
1.2 Scope of Work

In this work we will consider two types of ROMs. First, we enhance the existing POD-TPWL ROM, and extend it to new reservoir simulation applications. Second, we develop a deep-learning-based ROM. In the context of reservoir simulation, the POD-TPWL procedure has been applied to oil-water systems [15], including well control optimization [14, 48] and history matching [49], thermal recovery processes [97], and compositional simulation and optimization [46]. However, the application of POD-TPWL to CO\textsubscript{2} storage optimization has not yet been studied. In addition, the use of POD-TPWL for coupled flow-geomechanics problems has not been considered.

Existing POD-TPWL implementations use training data from 3-5 training simulations, and they typically linearize around a single training run. Incorporating training data from a large number (e.g., hundreds) of simulation runs in the offline procedure, which is potentially required for global optimization, where potential solutions can be anywhere in the search space, remains a challenge for POD-TPWL. The development of ROMs based on deep-learning techniques could address this issue and thus enhance the robustness of the model. Such a ROM, which could accommodate large variations in control variables, has yet to be investigated.

Towards these goals, in this work we extend POD-TPWL to treat coupled flow-geomechanics and optimization problems in CO\textsubscript{2} storage operations. Several enhancements are introduced to improve performance. We also develop a deep-learning-based ROM, specifically embed-to-control (E2C), and apply it to an oil-water problem. The goals of this work are:

- To extend POD-TPWL for application to CO\textsubscript{2} storage problems. This part of the work extends the author’s Master’s research [58]. The extensions developed here include the modification of the basic method to handle linearization around different training runs for improved POD-TPWL accuracy, and application of the resulting methodology to optimization problems involving the minimization of a risk measure. The states and derivatives required by our POD-TPWL methodology are provided by Stanford’s Automatic-Differentiation-based General Purpose Research Simulator, AD-GPRS [126].
• To develop a POD-TPWL ROM applicable for coupled flow-geomechanics problems. This requires the generalization of essentially all aspects of the POD-TPWL formulation. The full-order simulator, AD-GPRS, is modified to provide the necessary flow and geomechanical outputs. Five different basis matrices are constructed (using POD) to enable low-dimensional representations of all flow and geomechanical variables. Linearization around multiple training runs is again a component of the method. A Mohr-Coulomb failure criterion based on POD-TPWL results is applied to provide fast forecasts of rock failure during reservoir production.

• To develop a deep-learning framework for reduced-order modeling of oil-water systems based on the E2C model [113] and physics-informed treatments [93, 128]. Two key modifications of the existing E2C model will be introduced. Specifically, we simplify the VAE to an AE to achieve better accuracy for (deterministic) test cases, and we incorporate a comprehensive loss function that introduces both PDE-based physical constraints and more strongly enforces consistency in well data. The latter treatment is important for improving the accuracy of well quantities of interest, which are essential in well control optimization procedures.

1.3 Dissertation Outline

This dissertation proceeds as follows. In Chapter 2, we develop a POD-TPWL ROM for handling coupled flow-geomechanics problems. The formulation involves two primary variables for the flow system and three primary variables for the geomechanical system. Separate POD basis matrices are constructed to represent these five variables. Results for 2D systems are then presented. These are followed by a systematic study of error, under different training strategies, for several quantities of interest over a wide range of test cases. POD-TPWL results for a prototype 3D system are also presented. The work in Chapter 2 was presented at the 2019 SPE Reservoir Simulation Conference [60] and has been accepted by SPE Journal. Oleg Volkov and Timur Garipov are co-authors on this work.

In Chapter 3, we extend the POD-TPWL framework for CO$_2$ storage operations
originally developed in the author’s Master’s research [58]. The new aspects presented here include linearization around multiple training runs (referred to as the use of multiple derivatives), application of the ROM to new aquifer models, and optimization involving the minimization of a risk metric. In this chapter, we describe the POD-TPWL formulation in the context of geologic CO$_2$ storage. Reduction in error using the multiple-derivative treatment is demonstrated for two aquifer systems. Finally, we apply POD-TPWL for MADS-based optimization. Note that we conducted the work in Chapter 3 prior to that in Chapter 2. Since the flow-geomechanics formulation is the more general, however, we reduce the amount of repetition by presenting it first. The work in Chapter 3 was published in *International Journal of Greenhouse Gas Control* [59] in 2018.

In Chapter 4, we develop a deep-learning-based ROM. The embed-to-control (E2C) formulation, which is based on an auto-encoder and a linear transition model, is applied to 2D oil-water problems. The E2C formulation is analogous to POD-TPWL in many aspects, and these connections are highlighted. A loss function that incorporates physical constraints and enforces well data consistency is introduced. We present results for well responses and field quantities for a particular example, and then provide detailed error statistics for 100 test cases. A paper on this work, which includes Yimin Liu as a co-author, was submitted to *Journal of Computational Physics* (currently available on ArXiv [61]).

In Chapter 5, we conclude this dissertation with a summary and suggestions for future research directions. The detailed architecture for the modules in the deep-learning-based ROM are provided in Appendix A. Additional simulation results with the E2C ROM are presented in Appendix B.
Chapter 2

POD-TPWL for Coupled Flow-Geomechanics

Coupled flow-geomechanics simulations are computationally expensive because of the nonlinear multiphysics nature of the governing equations, and because the domain over which the geomechanical problem must be solved is typically much larger than that for the flow problem. Thus the application of reduced-order modeling will be useful in these problems. Our goal in this work is to extend the POD-TPWL framework for application to coupled flow-geomechanics systems. The method is applied to simulate production (via water injection) in heterogeneous reservoirs, with the geomechanical response of the formation included in the model. Key quantities of interest include well production and injection rates and the maximum and minimum principal stress fields.

The flow component of the problem considered here includes oil and water phases. The new POD-TPWL framework requires generalization of essentially all aspects of the formulation, including modification of the full-order simulator AD-GPRS [126] to provide the necessary flow and geomechanical outputs, and the construction of five separate basis matrices (using POD) to enable low-dimensional representations of all flow and geomechanical variables. We apply the ‘multiple derivatives’ procedure in this work, meaning that during test-case simulations, we linearize around solutions from different training runs, not just different time steps within a single training run. We use the Mohr-Coulomb failure criterion to determine the location and time for
rock failure, which is an important prediction provided by coupled flow-geomechanics simulations.

2.1 POD-TPWL Formulation

In this section, we introduce the POD-TPWL formulation for coupled flow and geomechanics systems. We first present the equations governing coupled oil-water flow and geomechanics. The POD-TPWL reduced-order model for this system is then described in detail. The POD-TPWL presentation here generally follows those of He et al. [48] and He and Durlofsky [47], though the details and implementation are quite different since we consider a multiphysics system.

2.1.1 Governing equations

We simulate coupled flow and geomechanics using the existing geomechanical module [34] in Stanford’s Automatic Differentiation-based General Purpose Research Simulator, AD-GPRS [126]. The governing equations for the flow system consist of mass conservation statements for the oil and water components (which are assumed to exist in immiscible oil and water phases), combined with Darcy’s law for each phase. The resulting equations are given by:

\[
\frac{\partial}{\partial t} (\phi S_j \rho_j) - \nabla \cdot [\lambda_j \rho_j k (\nabla p_j + \rho_j g \nabla D)] + \sum_w q_j^w = 0, \tag{2.1}
\]

where \(t\) denotes time, \(\phi\) indicates porosity, and the subscript \(j\) \((j = o, w)\) designates fluid phase \((o\) for oil and \(w\) for water). The variable \(S_j\) indicates phase saturation, \(\rho_j\) denotes phase density, \(\lambda_j\) is phase mobility \((\lambda_j = k_{rj}/\mu_j)\), where \(k_{rj}\) indicates relative permeability of phase \(j\) and \(\mu_j\) is the viscosity of phase \(j\), and \(q_j^w\) designates the phase source/sink term for well \(w\). Other variables are the permeability tensor \(k\), phase pressure \(p_j\), gravitational acceleration \(g\), and depth \(D\).

In the coupled flow-geomechanics model, the porosity \(\phi\) depends on the fluid pressure and the deformation of the solid skeleton. This relationship can be represented...
by:

$$
\phi = \phi_0 + \frac{(\alpha - \phi_0)(1 - \alpha)}{K_d}(p_f - p_{f,0}) + \alpha(\epsilon_\nu - \epsilon_{\nu,0}),
$$

(2.2)

where $\alpha$ is the Biot coefficient, $K_d$ denotes the drainage bulk modulus, $p_f$ is the average fluid pressure, which is determined by $p_f = \sum_j p_j s_j$, $\epsilon_\nu$ designates the volumetric strain, and the subscript 0 denotes the reference state. Note that in our implementation permeability is not affected by changes in porosity.

For the geomechanical system, the conservation of momentum equation for the solid skeleton can be written as:

$$
\nabla \cdot \sigma + \rho_b g \nabla D = 0,
$$

(2.3)

where $D$ is again depth and $\sigma$ is the total stress tensor, which contains contributions from both the fluid and solid skeleton of the rock. The bulk density of the saturated rock, $\rho_b$, can be represented in terms of fluid density $\rho_f$ and the solid skeleton density $\rho_s$ using

$$
\rho_b = \phi \rho_f + (1 - \phi) \rho_s.
$$

(2.4)

Eq. 2.3 is a quasi-static representation of reservoir deformation. Dynamic terms are neglected, as is common in the modeling of slow deformation processes. The stress on the solid skeleton, which is referred to as the effective stress and is designated by $\sigma'$, is a key quantity of interest in this study. This quantity is related to the total stress $\sigma$ via the following relationship:

$$
\sigma = \sigma' - \alpha p_f 1,
$$

(2.5)

where $1$ is the second-order identity tensor. The relationship between total stress and strain is given by the constitutive relation:

$$
\sigma = \mathbb{C} : \epsilon - \alpha p_f 1,
$$

(2.6)

where $\mathbb{C} = \mathbb{C}(E, \nu)$ is the fourth-order stiffness tensor of the solid skeleton, which is a function of Young’s modulus $E$ and Poisson’s ratio $\nu$. The symmetric second-order
2.1. POD-TPWL FORMULATION

The strain tensor $\epsilon$ is computed from the displacement vector $d$ using

$$\epsilon = \frac{1}{2}(\nabla d + \nabla^T d), \quad (2.7)$$

where the superscript $T$ denotes transpose, and $\nabla^T d$ is used to denote $(\nabla d)^T$.

The oil and water flow equations are discretized using a standard finite volume (FV) formulation, and their solutions are computed for each grid block (or FV cell). The geomechanical equations are discretized using a finite element (FE) formulation, and solutions are computed at each node. In a standard logically-Cartesian grid system without faults or fractures, the FE nodes map directly to the vertices of FV cells. We use $n_b$ and $n_d$ to denote the number of FV grid blocks and FE nodes, respectively, in the coupled system.

The oil-water model described in Eq. 2.1 is completed by enforcing the saturation constraint ($S_o + S_w = 1$) and by specifying a capillary pressure relationship $p_c(S_w) = p_o - p_w$. In this work, we take the capillary pressure to be zero. The flow system is fully defined through the use of two primary variables, $p = p_o = p_w$ and $S_w$, in each grid block. The geomechanical system is defined by the set of primary variables $d = [d_x, d_y, d_z]^T$ at each FE node. The total number of variables in the system is thus $n_v = 2n_b + 3n_d$.

We define $x_f = [(p, S_w)_1, \ldots, (p, S_w)_{n_b}]^T \in \mathbb{R}^{2n_b}$ to be the state vector for the flow variables, and $x_m = [(d_x, d_y, d_z)_1, \ldots, (d_x, d_y, d_z)_{n_d}]^T \in \mathbb{R}^{3n_d}$ to be the state vector for the geomechanical variables. The set of nonlinear algebraic equations representing the discretized fully-implicit system can be expressed as:

$$g(x^{n+1}, x^n, u^{n+1}) = 0, \quad (2.8)$$

where $g \in \mathbb{R}^{n_v}$ is the residual vector (set of nonlinear algebraic equations) we seek to drive to zero, $x = [x_f^T, x_m^T]^T \in \mathbb{R}^{n_v}$ represents the system states ($p$ and $S_w$ in each FV cell, $d_x, d_y, d_z$ at each FE node), $n$ indicates the previous time level and $n + 1$ the next time level, and $u \in \mathbb{R}^{n_u}$ designates the well control variables (bottom-hole pressure or BHP in our examples, though well rates could also be used), where $n_u$ is the number of wells in the system. When an actual simulation is performed, $x^n$ is known and $u^{n+1}$ is specified, and the goal is to solve for $x^{n+1}$. Equations of this
form have appeared in many of the previous POD-TPWL formulations, though the systems considered did not include geomechanics.

The full-order discretized nonlinear system defined by Eq. 2.8 is solved using Newton’s method. This requires constructing the sparse Jacobian matrix of dimension \( n_v \times n_v \), and then solving a linear system of dimension \( n_v \), at each iteration for every time step. These linear solutions are often the most time-consuming part of the simulation. As we will see, POD-TPWL avoids the construction of this high-dimensional \( n_v \times n_v \) system, and it involves only linear solutions.

The coupled flow-geomechanics system described above can be solved in either a loosely coupled or a fully coupled manner (there are many variants within these two categories). In a loosely coupled approach, the idea is to advance the flow problem and then solve the geomechanical problem at particular time steps. Such procedures may be adequate in some settings, but they can lose accuracy for problems with important couplings. Garipov et al. [34], for example, modeled a heavy oil (bitumen) reservoir producing under steam-assisted gravity drainage, in which the Young’s modulus of the reservoir was two orders of magnitude smaller than that of the non-pay zone. They showed that a loosely (one-way) coupled model could lead to a 20% error in reservoir pressure estimation. Such an approach is inadequate for accurately predicting rock failure, which is a key quantity of interest in this study.

Our POD-TPWL procedure could be applied for either loosely or fully coupled models. In the former case, we essentially treat the geomechanical problem separately from the flow problem. Here we consider the (more computationally challenging) fully coupled approach. This means that at every time step the full flow-geomechanics problem is solved. Fully coupled problems themselves can be addressed in different ways, and in this work we apply a fully implicit treatment.

It is important to emphasize that our reduced-order POD-TPWL model is constructed based on a particular forward simulator (AD-GPRS), and that we ‘inherit’ the capabilities and limitations of this simulator. This is the case with essentially all ROMs – the physics they are able to represent is dictated by the simulator on which they are based. The current capabilities of AD-GPRS include flow and geomechanically induced deformation. AD-GPRS does not, however, currently model dynamic fracture propagation, as would be important, for example, in simulating a
hydraulic fracturing operation. Thus our POD-TPWL model will not be able to simulate such processes either. It is possible that some types of models that do predict fracturing, such as phase-field models [80], could be represented using ROMs such as POD-TPWL. Although this is an interesting and important topic, it is beyond the scope of this work. We note finally that our POD-TPWL ROM is not restricted to regular grids, and it is applicable for models containing existing (static) faults.

2.1.2 Linearized representation

In the POD-TPWL procedure, a small number of training simulations (three or five in this work) are performed with specified well controls, and solutions and derivative matrices are generated and saved. The computed states are used to construct basis matrices, and these matrices are applied both for the low-dimensional representation of solution states and for constraint reduction. Subsequently, when a new (test) simulation involving a different set of controls is performed, low-dimensional solutions are computed using a linearized representation. Physical variables can be reconstructed, as needed, from the POD basis matrices.

We first describe the linearization (TPWL) process. In this development, the superscripts $i$ and $i+1$ designate consecutive time steps in a particular training simulation, and $n$ and $n+1$ indicate consecutive time steps in the test simulation. Given a test-run solution at time step $n$, the residual vector for the test run at time step $n+1$ can be represented by the following first-order Taylor series expansion:

$$g^{n+1} = 0 \approx g^{i+1} + \frac{\partial g^{i+1}}{\partial x^{i+1}}(x^{n+1} - x^{i+1}) + \frac{\partial g^{i+1}}{\partial x^i}(x^n - x^i) + \frac{\partial g^{i+1}}{\partial u^{i+1}}(u^{n+1} - u^{i+1}), \quad (2.9)$$

where $g^{i+1} = g(x^{i+1}, x^i, u^{i+1}) = 0$ since this is a training solution, and $g^{n+1} = g(x^{n+1}, x^n, u^{n+1})$ is also set to 0 since this is the residual we seek to drive to zero. After rearrangement, Eq. 2.9 can be written as:

$$J^{i+1}(x^{n+1} - x^{i+1}) = -[A^{i+1}(x^n - x^i) + B^{i+1}(u^{n+1} - u^{i+1})], \quad (2.10)$$
where the three matrices are defined as:

\[
J_{i+1} = \frac{\partial g_{i+1}}{\partial x_{i+1}} \in \mathbb{R}^{n_v \times n_v}, \quad A_{i+1} = \frac{\partial g_{i+1}}{\partial x_{i+1}} \in \mathbb{R}^{n_v \times n_v}, \quad B_{i+1} = \frac{\partial g_{i+1}}{\partial u_{i+1}} \in \mathbb{R}^{n_v \times n_u}.
\]

Note that \( J_{i+1} \in \mathbb{R}^{n_v \times n_v} \) is the Jacobian matrix at time step \( i + 1 \) of the training simulation, evaluated for the converged system.

The only unknown in Eq. 2.10 is \( x_{n+1} \), and this can be computed without any nonlinear iteration once the control vector \( u^{n+1} \) is prescribed. However, in order to improve computational efficiency, we will project the equations and unknowns into a lower-dimensional subspace. This is accomplished by representing the states using basis matrices constructed from POD, and by using LSPG to project the system of equations.

### 2.1.3 POD and constraint reduction

The state reduction procedure entails the use of proper orthogonal decomposition to construct a basis matrix \( \Phi \in \mathbb{R}^{n_v \times l} \) that enables the representation of state variables \( x \) in terms of a reduced state vector \( \xi \in \mathbb{R}^l \), where \( l (l << n_v) \) is the dimension of the subspace. Specifically, we write

\[
x \approx \Phi \xi. \quad (2.12)
\]

The basis matrix \( \Phi \), which will be described in detail below, is constructed by appropriately combining \( \Phi_f \) (basis matrix for flow variables) and \( \Phi_m \) (basis matrix for geomechanical variables). The matrices \( \Phi_f \) and \( \Phi_m \) are obtained by performing singular value decomposition (SVD) of the ‘snapshot’ matrices, which are matrices containing, as their columns, converged states generated during training runs.

For the flow states, the procedure is performed separately for the two state variables \( p \) and \( S_w \). The states determined in all of the full-order training simulations are entered as columns in the two state matrices \( X_p \in \mathbb{R}^{n_b \times L} \) and \( X_S \in \mathbb{R}^{n_b \times L} \), where \( L \) is the total number of snapshots generated during all of the training simulations. The solution at each time step provides a snapshot. We express \( X_p \) and \( X_S \) as

\[
X_p = [p^1, p^2, \cdots, p^L], \quad X_S = [S_w^1, S_w^2, \cdots, S_w^L].
\]

(2.13)
2.1. POD-TPWL FORMULATION

where the vectors \( \mathbf{p}^i \in \mathbb{R}^{n_b} \) and \( \mathbf{S}_w^i \in \mathbb{R}^{n_b} \), \( i = 1, \ldots, L \), indicate the pressure and water saturation in each of the \( n_b \) grid blocks at a particular time step in a training run. Similarly, for the geomechanical states, we construct

\[
\mathbf{X}_{d_x} = [\mathbf{d}_x^1, \mathbf{d}_x^2, \ldots, \mathbf{d}_x^L] \in \mathbb{R}^{n_d \times L}, \quad \mathbf{X}_{d_y} = [\mathbf{d}_y^1, \mathbf{d}_y^2, \ldots, \mathbf{d}_y^L] \in \mathbb{R}^{n_d \times L}, \quad \mathbf{X}_{d_z} = [\mathbf{d}_z^1, \mathbf{d}_z^2, \ldots, \mathbf{d}_z^L] \in \mathbb{R}^{n_d \times L},
\]

(2.14)

where the vectors \( \mathbf{d}_x^i \in \mathbb{R}^{n_d} \), \( \mathbf{d}_y^i \in \mathbb{R}^{n_d} \), and \( \mathbf{d}_z^i \in \mathbb{R}^{n_d} \), \( i = 1, \ldots, L \), indicate the displacements at each of the \( n_d \) nodes at a particular time step in a training run.

After performing SVD separately for \( \mathbf{X}_p \) and \( \mathbf{X}_S \), we retain a limited number of left singular vectors, which comprise the columns of the basis matrices \( \Phi_p \) and \( \Phi_S \). The number of left singular vectors in the two matrices, denoted as \( l_p \) and \( l_S \), are determined (independently) based on an energy criterion or on limited numerical experimentation [47]. In general, POD-TPWL results are not very sensitive to \( l_p \) and \( l_S \) (once these values exceed a minimum threshold), so this determination is not difficult.

Similarly, \( \Phi_{d_x} \), \( \Phi_{d_y} \), and \( \Phi_{d_z} \) are obtained by performing (separate) SVD of \( \mathbf{X}_{d_x} \), \( \mathbf{X}_{d_y} \), and \( \mathbf{X}_{d_z} \), with \( l_{d_x} \), \( l_{d_y} \), and \( l_{d_z} \) denoting the number of columns retained. Combining these matrices, we can write the basis matrix for the flow variables \( \Phi_f \), and the basis matrix for the geomechanical variables \( \Phi_m \), as

\[
\Phi_f = \begin{bmatrix} \Phi_p & 0 \\ 0 & \Phi_S \end{bmatrix}, \quad \Phi_m = \begin{bmatrix} \Phi_{d_x} & 0 & 0 \\ 0 & \Phi_{d_y} & 0 \\ 0 & 0 & \Phi_{d_z} \end{bmatrix},
\]

(2.15)

where \( \Phi_p \in \mathbb{R}^{n_b \times l_p} \), \( \Phi_S \in \mathbb{R}^{n_b \times l_S} \), \( \Phi_{d_x} \in \mathbb{R}^{n_d \times l_{d_x}} \), \( \Phi_{d_y} \in \mathbb{R}^{n_d \times l_{d_y}} \), and \( \Phi_{d_z} \in \mathbb{R}^{n_d \times l_{d_z}} \).

Defining \( l_f = l_p + l_S \) and \( l_m = l_{d_x} + l_{d_y} + l_{d_z} \), we have \( \Phi_f \in \mathbb{R}^{2n_b \times l_f} \) and \( \Phi_m \in \mathbb{R}^{3n_d \times l_m} \). Note that the block-diagonal representation in Eq. 2.15 is conceptual, as elements of these matrices are interspersed in the actual implementation. This will be explained in more detail later.

The full-order flow and geomechanical states can now be expressed in terms of
CHAPTER 2. POD-TPWL FOR COUPLED FLOW-GEOMECHANICS

low-dimensional variables as follows:

\[ x = \begin{bmatrix} x_f \\ x_m \end{bmatrix} \approx \Phi \xi = \begin{bmatrix} \Phi_f & 0 \\ 0 & \Phi_m \end{bmatrix} \begin{bmatrix} \xi_f \\ \xi_m \end{bmatrix}. \tag{2.16} \]

Here \( \xi_f \in \mathbb{R}^{l_f} \) and \( \xi_m \in \mathbb{R}^{l_m} \) are the reduced variables for the flow and geomechanical states. Defining \( l = l_f + l_m \), Eq. 2.16 enables us to express the \( n_v = 2n_b + 3n_d \) primary unknowns in terms of \( l \) variables, with \( l \ll n_v \).

We now return to the linearized representation for \( x^{n+1} \) presented in Eq. 2.10. Introducing Eq. 2.16 into Eq. 2.10 we obtain:

\[ J^{i+1} \Phi(\xi^{n+1} - \xi^{i+1}) = -[A^{i+1} \Phi(\xi^n - \xi^i) + B^{i+1}(u^{n+1} - u^{i+1})]. \tag{2.17} \]

Eq. 2.17 represents an over-determined system involving \( n_v \) equations in only \( l \) variables. To render the system well-posed, we need to project it into an \( l \)-dimensional subspace. This step is referred to as constraint reduction or projection. The projection matrix is denoted \( \Psi \in \mathbb{R}^{n_v \times l} \), and constraint reduction entails left-multiplying Eq. 2.17 by \( \Psi^T \). The low-dimensional linearized system of equations is now given by:

\[ \Psi^T J^{i+1} \Phi(\xi^{n+1} - \xi^{i+1}) = -\Psi^T[A^{i+1} \Phi(\xi^n - \xi^i) + B^{i+1}(u^{n+1} - u^{i+1})], \tag{2.18} \]

which involves \( l \) equations and \( l \) unknowns. Eq. 2.18 can be rearranged to give:

\[ \xi^{n+1} = \xi^{i+1} - (J_r^{i+1})^{-1}[A_r^{i+1}(\xi^n - \xi^i) + B_r^{i+1}(u^{n+1} - u^{i+1})], \tag{2.19} \]

where the reduced derivative matrices are given by

\[ J_r^{i+1} = (\Psi^{i+1})^T J^{i+1} \Phi, \quad A_r^{i+1} = (\Psi^{i+1})^T A^{i+1} \Phi, \quad B_r^{i+1} = (\Psi^{i+1})^T B^{i+1}. \tag{2.20} \]

Here, \( J_r^{i+1} \in \mathbb{R}^{l \times l}, A_r^{i+1} \in \mathbb{R}^{l \times l}, \) and \( B_r^{i+1} \in \mathbb{R}^{l \times n_u} \). Because there are many more grid blocks and nodes than wells in reservoir simulation problems, we have \( n_u \ll n_b \) and \( n_a \ll n_d \).

The two widely used constraint reduction procedures are Galerkin projection, in which \( \Psi^{i+1} = \Phi \) (with \( \Phi \) the basis matrix for the state variables defined in Eq. 2.16),
2.1. POD-TPWL FORMULATION

and the least-squares Petrov-Galerkin (LSPG) procedure, where $\Psi^{i+1} = J^{i+1} \Phi$. Detailed assessments of these two treatments are provided by He and Durlofsky [47] (for POD-TPWL) and Carlberg et al. [17] (for GNAT). It is clearly shown in both papers that, for many problems, Galerkin projection can lead to numerical instability in the ROM. LSPG, by contrast, was demonstrated to yield much more stable ROMs. For this reason, LSPG will be used in this work, as well as in the CO$_2$ storage study in Chapter 3.

2.1.4 Detailed numerical treatments

We now describe some important aspects of our POD-TPWL implementation. These include the assembly of the basis matrices, the structure of AD-GPRS Jacobian sub-matrices, and the POD-TPWL point selection scheme (which defines the training run and time step around which we linearize). These issues are now considered in turn.

The block-diagonal structure of $\Phi_f$ shown in Eq. 2.15 applies if the flow state unknowns $x_f$ are ordered with pressure in all blocks followed by water saturation in all blocks (i.e., $p_1, \ldots, p_{n_b}, S_{w,1}, \ldots, S_{w,n_b}$). In our implementation, however, the flow state unknowns $x_f$ are ordered as $(p_1, S_{w,1}), \ldots, (p_{n_b}, S_{w,n_b})$. Therefore, components of $\Phi_p$ and $\Phi_S$ appear interspersed within $\Phi_f$, as follows:

$$x_f = \begin{bmatrix} p_1 \\ S_{w,1} \\ \vdots \\ p_{n_b} \\ S_{w,n_b} \end{bmatrix} \approx \Phi_f \xi_f = \begin{bmatrix} \phi_{p,1} & \cdots & \phi_{p,l_p} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \phi_{S_{w,1}}^1 & \cdots & \phi_{S_{w,1}}^{l_S} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \phi_{p,n_b}^1 & \cdots & \phi_{p,n_b}^{l_p} & 0 & \cdots & 0 \\ 0 & \cdots & 0 & \phi_{S_{w,n_b}}^1 & \cdots & \phi_{S_{w,n_b}}^{l_S} \end{bmatrix} \begin{bmatrix} \xi_{p,1} \\ \vdots \\ \xi_{p,l_p} \\ \xi_{S_{w,1}} \\ \vdots \\ \xi_{S_{w,l_S}} \end{bmatrix}, \quad (2.21)$$

where $\phi_{p,i}^j, i = 1, \ldots, n_b, j = 1, \ldots, l_p$ and $\phi_{S_{w,i}}^j, i = 1, \ldots, n_b, j = 1, \ldots, l_S$, are components of $\Phi_p$ and $\Phi_S$, and $\xi_{p,j}, j = 1, \ldots, l_p$ and $\xi_{S_{w,j}}, j = 1, \ldots, l_S$ are the entries in the reduced flow state vector $\xi_f$. More details regarding the ordering of $\Phi_f$ can be found in Cardoso et al. [16]. In similar fashion, the geomechanical unknowns $x_m$ are ordered as $(d_{x_1}, d_{y_1}, d_{z_1}), \ldots, (d_{x_{n_b}}, d_{y_{n_b}}, d_{z_{n_b}})$. Thus, components of $\Phi_{dx}, \Phi_{dy}$ and $\Phi_{dz}$ are interspersed within $\Phi_m$, in a manner analogous to that shown.
in Eq. 2.21. This ordering, and the general structure of \( \Phi_f \) and \( \Phi_m \), are used to maintain compatibility with the Jacobian matrices generated by AD-GPRS.

We now describe the correspondence between AD-GPRS Jacobian submatrices and the matrices appearing in our POD-TPWL formulation. In the geomechanical formulation in AD-GPRS, the governing equations are ordered in the following way: (1) flow equations, (2) well equations, and (3) geomechanical equations. The same ordering also applies to the corresponding primary variables. Therefore, at a particular training-run time step \( i + 1 \), the ‘extended’ Jacobian matrix \( J^{i+1}_{\text{ext}} \) output from AD-GPRS can be expressed as

\[
J^{i+1}_{\text{ext}} = \begin{bmatrix}
J_{FF} & J_{FW} & J_{FM} \\
J_{WF} & J_{WW} & J_{WM} \\
J_{MF} & J_{MW} & J_{MM}
\end{bmatrix}^{i+1} \in \mathbb{R}^{(n_v+n_u) \times (n_v+n_u)}.
\] (2.22)

Here the first subscript of each submatrix indicates the corresponding equations (with F denoting flow, W well, and M geomechanics), and the second subscript indicates the corresponding variables. Note that the components in each submatrix of \( J^{i+1}_{\text{ext}} \) are interspersed in a manner analogous to that shown in Eq. 2.21. We extract the derivatives of the flow and geomechanical (reservoir) equations with respect to reservoir variables to construct the matrix \( J^{i+1} \in \mathbb{R}^{n_v \times n_v} \) that appears in Eq. 2.11:

\[
J^{i+1} = \begin{bmatrix}
J_{FF} & J_{FM} \\
J_{MF} & J_{MM}
\end{bmatrix}^{i+1} \in \mathbb{R}^{n_v \times n_v}.
\] (2.23)

The matrix \( B^{i+1} \in \mathbb{R}^{n_v \times n_u} \) in Eq. 2.11 is formed by extracting the derivatives of the reservoir equations with respect to the well variables (BHPs in this case):

\[
B^{i+1} = \begin{bmatrix}
J_{FW} \\
J_{MW}
\end{bmatrix}^{i+1} \in \mathbb{R}^{n_v \times n_u}.
\] (2.24)

We note finally that the matrix \( A^{i+1} \in \mathbb{R}^{n_v \times n_v} \) (which contains previous-time information) in Eq. 2.11 is output separately.

The POD-TPWL procedure requires a point selection scheme to determine the training-run solution (and controls) around which to linearize. We describe a point
2.1. POD-TPWL FORMULATION

selection procedure in more detail in Section 3.1.4 and in [59] (as noted in Chapter 1, the work in Chapter 3 was actually performed prior to the work in this chapter), that enables POD-TPWL to linearize around any solution from any of the training runs. This approach is referred to as point selection using multiple derivatives. Most of the previous POD-TPWL formulations used only one training run for linearization (the so-called primary training run) – the other training runs were used only to provide solution snapshots.

Our goal in point selection is to find the training solution that is ‘closest’ to the current test solution. We denote $d = d(n, \tau, j)$ to be the distance (defined below) between the test solution and any training solution. Here $n$ denotes the time step in the test run, $\tau$ indicates the particular training run, and $j$ is the time step in training run $\tau$. The point in any training run that is the closest to the current test solution (this training solution is denoted by $i$ in Eq. 2.10) is determined by finding the $\tau$ and $j$ that minimize $d$. We denote this particular $(\tau, j)$ pair as $(\tau^*, j^*)$.

Various definitions for $d$ have been used, and the ideal choice may be somewhat problem dependent (thus some amount of numerical experimentation is useful). Previous studies have used weighted combinations of the relative differences in pore volume injected (or an analogous quantity) and in the reduced states [46, 59]. Here, after limited numerical experimentation, and following the procedure described in Section 3.1.4, we use an approach that involves only estimates of cumulative well-by-well injection and production. Specifically, we define $d(n, \tau, j)$ as

\begin{equation}
    d(n, \tau, j) = \sum_{k=1}^{n_w} \frac{|Q_{k}^{TPWL,n} - Q_{k}^{\tau,j}|}{Q_{k}^{TPWL,n} + \epsilon},
\end{equation}

where $n_w$ is the number of wells, $Q_{k}^{TPWL,n}$ is the estimated cumulative injection or production up to time step $n$ for well $k$ in the POD-TPWL model, $Q_{k}^{\tau,j}$ is the estimated cumulative injection or production in training run $\tau$ up to time step $j$ for well $k$, and $\epsilon = 0.001$ is used to prevent dividing by a very small number.

The quantity $Q_{k}^{TPWL,n}$, for single-block well completions, is given by

\begin{equation}
    Q_{k}^{TPWL,n} = \int_{0}^{t^n} d_{k}^{TPWL,n} dt,
\end{equation}
where \( t^n \) is the time associated with time step \( n \) in the test run and \( q_k^{\text{TPWL},n} \) denotes the estimated injection or production rate at time step \( n \). This quantity is given by

\[
q_k^{\text{TPWL},n} = W I_{b,k}(\lambda_{b,k}^n)(p_{b,k}^n - p_{w,k}^n),
\]

(2.27)

where \( W I_{b,k} \) is the Peaceman well index linking well \( k \) to block \( b \) (the block in which well \( k \) is completed), \( \lambda_{b,k}^n \) is the total mobility evaluated in the well block at time step \( n \), \( p_{b,k}^n \) is the block pressure for the block containing well \( k \), and \( p_{w,k}^n \) is the wellbore pressure for well \( k \) at time step \( n \). If well \( k \) is completed in multiple blocks, Eq. 2.27 would be extended to include a sum over all blocks \( b \) in which the well is open to flow. An analogous expression is applied to compute \( Q_{\tau,j}^{*} \).

In our actual implementation, we do not search over all time steps in all training runs to find \((\tau^{*}, j^{*})\), but rather over a limited set, as in Section 3.1.4. Also, we do not include the well-block mobility terms in our evaluations of \( Q_k^{\text{TPWL},n} \) and \( Q_{\tau,j}^{*} \). For injection wells, the well blocks quickly reach the maximum water saturation, so these terms are essentially the same in both the \( Q_k^{\text{TPWL},n} \) and \( Q_{\tau,j}^{*} \) expressions and thus cancel. For production wells there is more variation in well-block saturation so strict cancellation may not occur, but the error introduced by this treatment is small.

As noted above, many previous POD-TPWL implementations involved the use of derivative matrices from only a single training run. The impact of using derivative matrices from multiple training runs will be assessed in Section 3.2 for the CO\(_2\) storage problem. There we will show that this approach can provide significant improvement in POD-TPWL accuracy relative to using derivatives from a single training run. This observation is consistent with the earlier findings of Rousset [96], who applied POD-TPWL for idealized thermal problems.

Accuracy improvement can also be achieved through use of a ROM that includes treatment of nonlinearities, such as POD-TPWQ [107]. This procedure was shown to provide consistently more accurate results than POD-TPWL for 2D oil-water problems. Its extension into a general-purpose simulation setting is however challenging since it requires third-order tensor (Hessian-type) terms that are not otherwise computed in forward simulations.
2.1.5 POD-TPWL workflow

The offline and online POD-TPWL procedures are shown in Algorithms 1 and 2. For the coupled flow-geomechanics problems considered in this chapter, we perform either three or five full-order training runs. The overall offline process entails computation equivalent to about five full-order simulations in the former case, and about eight full-order simulations in the latter case. As is the case with most ROMs, the use of POD-TPWL is only appropriate if a significant number of test runs are to be performed.

At each time step in the online POD-TPWL computations, the training states \( i \) and \( i + 1 \) are determined through minimization of distance \( d(n, \tau, j) \). Then \( \xi^{n+1} \) is found by solving Eq. 2.19. The full-order solution can be reconstructed by applying \( x^{n+1} = \Phi \xi^{n+1} \) as required. At most time steps the full-order solution is needed only at well blocks, where production, injection and rock-failure quantities are computed. The production and injection rates are calculated through application of Eq. 2.27. The computation of rock-failure quantities will be described in the following section. In some of the results below we present the global solution, constructed using \( x^{n+1} = \Phi \xi^{n+1} \), at selected time steps.

Algorithm 1 POD-TPWL offline procedure

1: Perform training runs with specified control settings;
2: Load snapshots \( X_p, X_s, X_{d_x}, X_{d_y}, X_{d_z} \) (Eqs. 2.13 and 2.14) and derivatives \( J^i, A^i, B^i \) (Eqs. 2.23 and 2.24) from each training run;
3: Construct basis matrices (Eq. 2.15) from snapshots of training runs;
4: Construct reduced states \( \xi^i \) (using \( \xi^i = \Phi^T x^i \)) and reduced derivatives \( J_r^i, A_r^i, B_r^i \) (Eq. 2.20) for all training runs;

2.2 POD-TPWL Simulation Results

We now demonstrate the performance of our POD-TPWL procedure for coupled flow-geomechanics problems. The procedure will be applied to a 2D vertical cross-sectional model and to a prototype 3D example. A systematic error study is performed for the 2D case.
Algorithm 2 POD-TPWL online procedure

1: for $n = 1$ to $n_f$ (final simulation time step) do
2: Find training state $\xi^i$ that is the closest to $\xi^n$ (using Eq. 2.25);
3: Solve Eq. 2.19 for $\xi^{n+1}$;
4: end for
5: Reconstruct full-order states $x^{n+1}$ from $\xi^{n+1}$ at required locations (e.g., well blocks) using Eq. 2.16;
6: Construct secondary variables (e.g., effective stresses using Eq. 2.5);
7: Compute well responses (Eq. 2.27) and rock-failure criterion (Eq. 2.30 below).

The flow system includes oil and water. The relative permeability functions are given by

\begin{align}
    k_{ro}(S_w) = k_{ro}^0 \left( \frac{1 - S_w - S_{or}}{1 - S_{wr} - S_{or}} \right)^a, \\
    k_{rw}(S_w) = k_{rw}^0 \left( \frac{S_w - S_{wr}}{1 - S_{wr} - S_{or}} \right)^b,
\end{align}

where $k_{ro}^0 = k_{rw}^0 = 1$, $S_{wr} = S_{or} = 0.2$, and $a = b = 2$. The densities are set to $\rho_o = 849 \text{ kg/m}^3$ and $\rho_w = 1025 \text{ kg/m}^3$. Capillary pressure effects are neglected.

### 2.2.1 Model 2.1: 2D example

The 2D cross-sectional ($x$-$z$) model is shown in Fig. 2.1. The reservoir model contains $220 \times 1 \times 85$ (total of 18,700) grid blocks and $221 \times 2 \times 86$ (total of 38,012) nodes. The grid blocks are of dimensions $10 \text{ ft} \times 8 \text{ ft} \times 5 \text{ ft}$. There are cap-rock and bedrock regions above and below the reservoir, as illustrated in Fig. 2.1(a). Zero-displacement boundary conditions are applied on the left, right, and lower boundaries of the model, while a constant stress of 900 bar is specified at the upper boundary. This value (900 bar) follows from assuming that the cap-rock is located 13,123 ft ($\sim 4000 \text{ m}$) below the surface, and that the average density of the overburden rock is around $2.25 \text{ g/cm}^3$. Note that this estimation based on average density is consistent with the approach described in Figures 1.3 and 1.4 in Zoback [129]. There are two injection wells and one production well in the model, as shown in Fig. 2.1(a).

Figure 2.1(b) displays the Gaussian log-permeability field. The correlation structure of the log-permeability field is characterized by an exponential variogram model, with a correlation length of 40 grid blocks in the $x$ direction and 5 grid blocks in the
2.2. POD-TPWL SIMULATION RESULTS

The arithmetic mean permeability of the reservoir rock ($\bar{k}$) is 42.3 mD, and the variance of log-permeability ($\sigma^2_{\log k}$) is 1.03. The permeability of the cap-rock and bedrock is six orders of magnitude lower than $\bar{k}$, i.e., $\bar{k}_{\text{cap}} = \bar{k}_{\text{bed}} = 48 \times 10^{-6}$ mD. In this model, we set $k_x = k_z$.

Figure 2.1(c) shows the porosity field of the model. The porosity ranges between 0.1 and 0.3. The correlation coefficient between permeability and porosity in the reservoir portion of the model is 0.75. There is essentially no flow in the bedrock and cap-rock due to the near-zero permeability in these regions, so the value of porosity is unimportant. Fig. 2.1(d) shows the layered Young’s modulus of the model. Following Baù et al. [7], Young’s modulus is taken to be a function of depth, and it is calculated based on the initial stress field of the reservoir. Poisson’s ratio is set to a constant value of 0.3.

The initial pressure at the top of the reservoir is 413 bar. This corresponds to the hydrostatic pressure at 13,248 ft (4038 m) below the surface (note that the top of the reservoir is 125 ft below the top of the cap-rock), with an average fluid density of 1.025 g/cm$^3$. The wells are all located in layer 43 of the model. Each of the wells is
completed in only a single grid block. The model is simulated for a total of 700 days. The full-order system contains $18,700 \times 2 + 38,012 \times 3 = 151,436$ primary variables.

The training and test-case BHP profiles for the two injection wells (I1 and I2) are shown in Fig. 2.2. Three training simulations with time-varying BHPs (shown as gray, blue and black lines) are performed to provide state and derivative information. The test-run BHP schedules for both wells are indicated by the red lines. BHPs in all runs are varied every 100 days. The BHP profile for the production well (P1) does not change between the training and test simulations. This BHP is set to 320 bar for the first 300 days and to 280 bar for the remaining 400 days. Note that, as discussed in many previous papers in this area, the time-varying training schedules should resemble, as closely as possible, the schedules expected during test runs. If the test schedules become very different from those used for training, as can occur, e.g., during production optimization procedures, retraining can be performed. This is relatively expensive, however, as it entails one or more additional full-order simulations.

A total of 87 snapshots are generated from the three full-order training simulations. The parameters $l_p$, $l_S$, $l_{dx}$, $l_{dy}$, and $l_{dz}$ are all set to 35 in this case. These values were determined by limited numerical experimentation, and we did not find the POD-TPWL results to be highly sensitive to this specification.

![Figure 2.2: Injection schedules for training and test runs (Model 2.1)](image)

We now assess the performance of POD-TPWL for this test case. Since the control parameters are time-varying BHPs for each of the wells, the (well) production
quantities of interest are phase flow rates, which are displayed in Fig. 2.3. The gray curves denote the results for the three training cases, the red curves correspond to the full-order (reference) AD-GPRS solution, and the blue curves represent the POD-TPWL results. It is evident that the POD-TPWL results are in consistently close agreement with reference AD-GPRS results for all quantities presented, which indicates that POD-TPWL is able to provide accurate predictions for production and injection rates for this case. Note that the POD-TPWL curves generally (but not always) fall within the range of the training results. We see, however, that the POD-TPWL results are closer to different training runs at different times. The use of multiple derivatives in point selection (i.e., the consideration of time steps in all training runs for linearization) leads to improved accuracy in such cases.

In addition to well rates, the global solution at particular times may also be of interest. In Fig. 2.4 we consider the global fluid pressure field at 400 days. Note that the vertical scale is stretched by about a factor of five, in this and subsequent figures for Model 2.1, to improve clarity. Fig. 2.4(a) displays the full-order pressure field (also referred to as the high-fidelity solution, HFS) for the test case, while Fig. 2.4(b) shows the pressure field at the same time for the POD-TPWL solution. The close visual agreement suggests that POD-TPWL is able to provide an accurate global pressure field.

The level of agreement between the two solutions is quantified in Fig. 2.4(d). Note the very different color bar scale here compared to that in Fig. 2.4(a) and (b). It is evident that the POD-TPWL pressure error is indeed very small, with the maximum error occurring at the boundaries between the reservoir and the cap-rock or bedrock. Error appears at these locations because pressure and other solution quantities vary strongly and nonlinearly in these regions, and POD-TPWL incurs some error (overshoot or undershoot) in such situations.

It is appropriate to consider the disparity between the test-run solution and the training solution used for linearization. This enables us to better assess the impact of the linearization applied in POD-TPWL. Fig. 2.4(c) shows the difference between the (HFS) training and test-case pressure fields. The training solution used here is the closest training point \((\tau^*, j^*)\) to the current test solution (at time \(n\)), which is
CHAPTER 2. POD-TPWL FOR COUPLED FLOW-GEOMECHANICS

(a) Water injection rate, Well I1

(b) Water injection rate, Well I2

(c) Oil production rate, Well P1

(d) Water production rate, Well P1

Figure 2.3: Well rates for test case (Model 2.1)
determined through

\[(\tau^*, j^*) = \arg \min_{\tau, j} d(n, \tau, j),\]  

(2.29)

where \(d\) is the distance defined in Eq. 2.25. Note that the training run containing
the closest training solution can change from time step to time step. The difference
between the test run and closest training solutions can be thought of as the error for
a ‘zeroth-order’ ROM; i.e., for such a ROM we would simply find the ‘closest’ training
solution at each time step and view this as representative of the test-case solution.
The improvement from applying POD-TPWL linearization is evident by comparing
Fig. 2.4(d) to Fig. 2.4(c). More specifically, except at the reservoir boundaries, the
error in Fig. 2.4(d) is about an order of magnitude less than the differences (zeroth-
order ROM errors) in Fig. 2.4(c). Note that in this and subsequent figures of this
type, the color bar scale in (d) extends over a much smaller range than that in (c).

Figure 2.5 displays analogous results for saturation at 400 days. The color scale
indicates oil saturation value (thus blue denotes water). We see that, at this time,
more closely all of the reservoir has been contacted by water. The POD-TPWL result (Fig. 2.5(b)) is again seen to be in close visual agreement with the high-fidelity AD-
GPRS solution (Fig. 2.5(a)). The error map in Fig. 2.5(d) further quantifies the
accuracy of the POD-TPWL solution. Note that, for saturation at 400 days, the
training and test-run solutions are quite close, as is evident from the difference map
in Fig. 2.5(c). Small differences are evident, however, around the saturation front.

There are also several geomechanical quantities of interest including global stress
fields. The maximum and minimum effective principal stress fields at 400 days are
shown in Figures 2.6 and 2.7. In both figures we see that the POD-TPWL results
closely match those from the full-order simulation. In addition, the errors in the POD-
TPWL test solution (Figures 2.6(d) and 2.7(d)) are significantly smaller than the
differences between the training and test solutions (Figures 2.6(c) and 2.7(c)). This
again demonstrates the effectiveness of the POD-TPWL treatment. Note finally that
the primary variables for geomechanics (displacements) are calculated at the nodes,
though the effective principal stresses displayed in these figures are calculated at grid
blocks within AD-GPRS. This treatment is also applied to POD-TPWL output.

The Mohr-Coulomb rock failure criterion characterizes the potential for shear
(a) High fidelity solution for test \( (\text{HFS}_{\text{test}}) \) (b) POD-TPWL solution for test \( (\text{TPWL}_{\text{test}}) \)

(c) \(|\text{HFS}_{\text{train}} - \text{HFS}_{\text{test}}|\) 

(d) \(|\text{TPWL}_{\text{test}} - \text{HFS}_{\text{test}}|\)

Figure 2.4: Pressure at 400 days for Model 2.1 (unit: bar)
2.2. **POD-TPWL SIMULATION RESULTS**

![Images](image1.png)

(a) \( HFS_{test} \)  
(b) \( TPWL_{test} \)  
(c) \( |HFS_{train} - HFS_{test}| \)  
(d) \( |TPWL_{test} - HFS_{test}| \)

**Figure 2.5**: Oil saturation at 400 days (Model 2.1)
Figure 2.6: Maximum effective principal stress ($\sigma_1'$) at 400 days for Model 2.1 (unit: bar)
2.2. POD-TPWL SIMULATION RESULTS

Figure 2.7: Minimum effective principal stress ($\sigma'_3$) at 400 days for Model 2.1 (unit: bar)
failure [106]. This quantity is described by the safety factor $\chi$, with

$$\chi = 1 - \frac{(\sigma'_1 - \sigma'_3)/2}{c \cos \phi + (\sigma'_1 + \sigma'_3)/\sin(\phi/2)} = 1 - \frac{\tau_m}{\tau^*_m}.$$  

(2.30)

Here $\sigma'_1$ and $\sigma'_3$ are the maximum and minimum principal components (i.e., maximum and minimum eigenvalues) of the effective stress tensor $\sigma'$ defined in Eq. 2.5. The variable $\tau_m = (\sigma'_1 - \sigma'_3)/2$ denotes the largest shear stress, while $\tau^*_m$ represents the maximum allowable shear stress. The variable $c$ is the internal cohesion (here we set $c = 0$) and $\phi$ is the angle of internal friction (here we set $\tan \phi = 0.65$). When $\chi$ reaches zero, shear failure is likely to occur.

We now assess the ability of POD-TPWL to predict this safety factor (which will also be referred to as the failure criterion). Results for $\chi$ at 400 days for the training run, HFS test case, and POD-TPWL test case are shown in Fig. 2.8. The color scales are the same in all three images. The rock failure threshold of $\chi < 0$ is not reached at any location in the training solution at 400 days (Fig. 2.8(a)). Rock failure does occur, however, at Well I1 in the full-order test solution (dark blue region near the left edge of Fig. 2.8(b)). Importantly, the POD-TPWL solution also captures this rock failure, as is evident in Fig. 2.8(c).

In this example, rock failure is most likely to occur at injection wells, where the fluid pressure is the highest. It is therefore of interest to assess the accuracy of the POD-TPWL estimate of $\chi$, through time, at injection well locations. These results are displayed in Fig. 2.9. We see that the POD-TPWL results (blue curves) are in close agreement with the full-order reference results (red curves) for both wells at nearly all times. Note in particular that, when $\chi$ passes through zero at around 400 days in Fig. 2.9(a), the POD-TPWL result tracks the AD-GPRS solution very closely.

The results for $\chi$ presented in Figures 2.8 and 2.9 enable us to estimate the time and location of rock failure. As discussed earlier, because the forward model on which our POD-TPWL representation is based does not describe fracture propagation, the ROM cannot provide predictions along these lines. Thus, after fracturing occurs, both the forward model and the POD-TPWL model will lose accuracy, especially in regions where $\chi < 0$. 
Figure 2.8: Mohr-Coulomb failure criterion ($\chi$) at 400 days (Model 2.1)
Since our POD-TPWL model can linearize around solutions from different training runs (three in this example), it is appropriate to assess how it behaves in this regard. Fig. 2.10 displays the selected training run for Model 2.1 over time. From the well BHP schedules shown in Fig. 2.2, we might expect the model to linearize around Training run 1 until 200 days, around Training run 3 from 200 days to 400 days, and around Training run 2 from 400 days until the end of the simulation. This is not exactly what occurs, however, as the point selection scheme tends to continue with a particular training run for somewhat longer than would be expected based on the BHP schedules. This is because the distance measure $d$ in Eq. 2.25, rather than proximity to the BHP schedule, is used for point selection. We performed limited comparisons between these two specifications, and found that the use of Eq. 2.25 provides better accuracy.

Finally, we discuss the timings for the high-fidelity and POD-TPWL runs. The typical time required for a full-order AD-GPRS simulation for this 2D example is around 2000 seconds. The online time required by POD-TPWL for this case is about 8 seconds. This corresponds to a runtime speedup of about 250. These timings (and thus the speedup observed) vary in practice depending on the condition of the computational cluster used for these runs. As noted earlier, the offline requirements for POD-TPWL model construction correspond to the time required for about five high-fidelity simulations. These pre-processing computations readily parallelize, so elapsed time would be closer to that for a single run if such resources are used.

2.3 Impact of Training Runs on POD-TPWL Accuracy

In the previous section, we demonstrated the detailed performance of POD-TPWL for a single test case. In this section, we conduct a systematic quantification of POD-TPWL error for a large number of test cases. The model here is the same 2D example as in the previous section. We apply different levels of perturbation to the training and test-case well schedules to assess the impact on POD-TPWL error. We also vary the POD-TPWL training by considering different numbers of training runs and by introducing randomness into the training schedules.
2.3. IMPACT OF TRAINING RUNS ON POD-TPWL ACCURACY

(a) Well I1
(b) Well I2

Figure 2.9: Mohr-Coulomb failure criterion ($\chi$) at the injection wells (Model 2.1)

Figure 2.10: Variation of selected training run with time (Model 2.1)
We again have two injection wells and one production well in the system and consider a simulation time frame of 700 days. The production well BHPs are the same for all training and test runs (320 bar for the first 300 days, 280 bar for the remaining 400 days), as in the previous section. We consider steadily increasing injection-well BHP profiles, as shown in Fig. 2.11. Three different levels of perturbation in the training injection-well schedules, corresponding to $\pm 5$, $\pm 10$ and $\pm 20$ bar, are considered. The detailed well specifications are shown in Fig. 2.11 (which corresponds to the $\pm 20$ bar case), where we see that Training run 1 entails BHPs that are 20 bar above those of Training run 2, while Training run 3 corresponds to BHPs that are 20 bar below those of Training run 2. We refer to this set of training runs as ‘3 regular trainings.’

The test-run BHPs are not constrained to lie strictly within the range defined by the training schedules. Rather, we extend this range by 25% to introduce more variability in the test cases. The test-run BHPs for I1 ($u_{test}^1$) are represented as

$$u_{test}^1 = u_{train,2}^1 + (1.25\gamma_{ptb})\alpha \tag{2.31}$$

where $u_{train,2}^1$ designates the BHP schedule for Well I1 in Training run 2, $\gamma_{ptb} \in [5, 10, 20]$ bar is the level of perturbation used in the training runs, and $\alpha \in \mathbb{R}^{n_{step}}$ is a random vector. Each component of $\alpha$ follows a uniform distribution, i.e., $\alpha_k \sim U(-1,1)$, $k = 1, \ldots, n_{step}$, with $n_{step}$ the number of control steps in the injection schedules. Here we set $n_{step} = 7$. The BHP schedule for Well I2 is constructed analogously. Test-case BHP profiles for Wells I1 and I2, generated in this manner, are shown in Fig. 2.11. Due to the factor of 1.25 in Eq. 2.31, these BHPs fall outside of the range of the training schedules at some control steps.

We also consider the use of two other types of training BHP schedules, as shown in Fig. 2.12. The procedure illustrated in Fig. 2.12(a) is referred to as ‘5 regular trainings.’ In this case, we perform two additional training runs, one with a BHP schedule that lies midway between those of Training runs 1 and 2 in the ‘3 regular trainings’ case, and one with a BHP schedule that lies midway between those of Training runs 2 and 3 in the ‘3 regular trainings’ case. The treatment shown in Fig. 2.12(b), referred to as ‘5 random trainings,’ involves two additional training runs with BHPs that are generated by randomly perturbing, by $\pm 20$ bar in this
2.3. IMPACT OF TRAINING RUNS ON POD-TPWL ACCURACY

To construct these additional BHP schedules we apply Eq. 2.31, but without the factor of 1.25 (e.g., \( u_{1,\text{train},4} = u_{1,\text{train},2} + \gamma_{\text{ptb}} \alpha \)). Test-run BHP profiles are constructed through application of Eq. 2.31, as described above.

![Training Schedules](image)

(a) Well I1  
(b) Well I2

Figure 2.11: Training schedules for ‘3 regular trainings’ procedure (\( \gamma_{\text{ptb}} = 20 \) bar), along with test-case schedule

![Alternative Training Schedules](image)

(a) 5 regular trainings  
(b) 5 random trainings

Figure 2.12: Alternative training schedules for Well I2 (\( \gamma_{\text{ptb}} = 20 \) bar)

We now define a number of error measures, which will be used to assess the performance of POD-TPWL for the different training procedures and values of \( \gamma_{\text{ptb}} \).
The oil and water production rate errors for a single well are defined as:

\[
e_o = \frac{\int_0^T |q_{TPWL}^o(t) - q_{HFS}^o(t)| \, dt}{\int_0^T |q_{HFS}^o(t)| \, dt} + \epsilon, \quad e_w = \frac{\int_0^T |q_{TPWL}^w(t) - q_{HFS}^w(t)| \, dt}{\int_0^T |q_{HFS}^w(t)| \, dt} + \epsilon,
\]

(2.32)

where \(q^o(t)\) and \(q^w(t)\) are the oil and water production rates at time \(t\), the subscripts HFS and TPWL denote the high-fidelity simulation (AD-GPRS) and POD-TPWL results, \(T\) is the total simulation time, and \(\epsilon = 0.001\) is a constant that prevents the denominator from being zero. Similarly, the injection rate error for a single well is defined as:

\[
e_{\text{inj}} = \frac{\int_0^T |q_{TPWL}^{\text{inj}}(t) - q_{HFS}^{\text{inj}}(t)| \, dt}{\int_0^T |q_{HFS}^{\text{inj}}(t)| \, dt} + \epsilon,
\]

(2.33)

where ‘inj’ denotes the water injection rate. We define the overall rate error \(E_r\) in terms of \(e_o\), \(e_w\) and \(e_{\text{inj}}\) for all wells:

\[
E_r = \frac{1}{n_p} \sum_{k=1}^{n_p} (e_o + e_w)_k + \frac{1}{n_i} \sum_{k=1}^{n_i} (e_{\text{inj}})_k,
\]

(2.34)

where \(n_p\) is the number of production wells and \(n_i\) is the number of injection wells.

The error in the well failure criterion is defined analogously as:

\[
E_x = \frac{1}{n_w} \sum_{k=1}^{n_w} \left[ \int_0^T |\chi_{TPWL}^k(t) - \chi_{HFS}^k(t)| \, dt \right] + \epsilon,
\]

(2.35)

where \(\chi^k(t)\) is the rock failure criterion at well \(k\) at time \(t\), and \(n_w\) is the total number of wells. In our case the wells are each perforated in a single grid block. If a well is completed in multiple blocks, the sum in Eq. 2.35 should extend over all blocks containing a well.
Error in global quantities is also of interest. We define global pressure and saturation error as:

\[
E_v = \frac{\sum_{k=1}^{n_b} \int_0^T \left| v_{TPWL}^k - v_{HFS}^k \right| dt}{\sum_{k=1}^{n_b} \int_0^T \left| v_{HFS}^k \right| dt + \epsilon},
\]

where \( v \) denotes either pressure \((p)\) or saturation \((S)\), and \( n_b \) is the total number of grid blocks in the model. The pressure and saturation errors \( E_p \) and \( E_S \) are then summed to give an overall flow state error \( E_{pS} \); i.e., \( E_{pS} = E_p + E_S \).

The error in the maximum effective principal stress field is defined analogously as:

\[
E_{\sigma_1} = \frac{\sum_{k=1}^{n_b} \int_0^T \left| \sigma_{1,TPWL}^{t,k} - \sigma_{1,HFS}^{t,k} \right| dt}{\sum_{k=1}^{n_b} \int_0^T \left| \sigma_{1,HFS}^{t,k} \right| dt + \epsilon}.
\]

The error in the minimum effective principal stress field \((E_{\sigma_3})\) is given by a similar expression. The errors in these two quantities are summed to give an overall effective principal stress error \( E_\sigma \); i.e., \( E_\sigma = E_{\sigma_1} + E_{\sigma_3} \).

We perform 20 test runs, with random BHP profiles generated using Eq. 2.31, for each level of perturbation \((\gamma_{ptb} = 5, 10 \text{ and } 20 \text{ bar})\) and for each of the three training procedures. This corresponds to a total of 180 different cases. Before discussing error statistics for these runs, we first present results for \( \chi(t) \) using different training procedures. The test run selected here is the median case, for \( \gamma_{ptb} = 20 \text{ bar} \), from the 20 test runs in the ‘3 regular trainings’ approach.

Results are displayed in Fig. 2.13, for Wells I1 and I2, using ‘3 regular trainings’ and ‘5 random trainings.’ This case involves a large perturbation, and there is clearly some error in the POD-TPWL results. The values of \( E_\chi \), which combine the errors for all wells, are \( E_\chi = 0.119 \) for ‘3 regular trainings’ and \( E_\chi = 0.067 \) for ‘5 random trainings.’ The differences between the two training procedures are evident by comparing Fig. 2.13(a) and (b), and Fig. 2.13(c) and (d). In the latter comparison, for example, we observe improvement in POD-TPWL accuracy between around 150 and 300 days, even though a degree of error clearly persists.

We next present error statistics, in the form of box plots, for overall rate error
Figure 2.13: Mohr-Coulomb failure criterion ($\chi$) for injection wells ($\gamma_{ptb} = 20$ bar). For ‘3 regular trainings’ $E_\chi = 0.119$, and for ‘5 random trainings’ $E_\chi = 0.067$.
2.3. IMPACT OF TRAINING RUNS ON POD-TPWL ACCURACY

($E_r$), well failure criterion error ($E_\chi$), global flow state error ($E_{ps}$), and global effective principal stress error ($E_p$). These results are displayed in Fig. 2.14. Each of the boxes represents the error distribution for 20 POD-TPWL test runs. In each box, the central red line indicates the median error, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The ‘whiskers’ extending out from the boxes indicate the minimum and maximum errors. The legend in Fig. 2.14(b) applies to all subfigures. In all plots the yellow boxes display the error for ‘3 regular trainings,’ the green boxes show the error for ‘5 regular trainings,’ and the blue boxes present the error for ‘5 random trainings.’

Recall that $E_{ps}$ and $E_\sigma$, presented in Fig. 2.14(c) and (d), involve averages over all grid blocks. However, the discrepancies between the POD-TPWL and full-order test solutions usually occur in only localized regions of the model (e.g., near the water front for $E_{S}$, which is a component of $E_{ps}$). Therefore, the magnitudes of these errors are small compared with those for $E_r$ and $E_\chi$.

Interestingly, even though the errors shown in Fig. 2.14 are for a wide range of quantities, consistent trends are evident. First, for a given training procedure, error increases as the level of perturbation ($\gamma_{ptb}$) increases. This is as expected since POD-TPWL involves only a (piecewise) linear treatment, and there are larger differences between training and test-case solutions as $\gamma_{ptb}$ increases. In addition, again consistent with expectations, we see that error levels are in all cases lower as we proceed from ‘3 regular trainings’ to ‘5 regular trainings.’ The additional training runs are beneficial because they yield more solution snapshots, and thus enriched basis matrices, and because they provide more training solutions around which linearization can be performed. Finally, we see that error is also consistently lower for ‘5 random trainings’ than it is for ‘5 regular trainings.’ This is an interesting result, and it will be useful to assess whether the inclusion of randomness in training schedules is advantageous for other cases modeled using POD-TPWL.

Finally, it is useful to estimate the level of perturbation associated with the BHP schedules in Fig. 2.2. Estimation of the approximate value of $\gamma_{ptb}$ for these profiles enables us to assess where the detailed results presented earlier fall in terms of the systematic assessment in this section. We estimate $\gamma_{ptb}$ for the test-case BHPs in Fig. 2.2 by first computing the time-average ‘distance’ between the test-case and
Figure 2.14: Systematic error assessment for Model 2.1. Legend in (b) applies to all subfigures.
nearest-training-run BHPs. We then find the value of $\gamma_{ptb}$ which, when used in Eq. 2.31, gives the same distance. Proceeding in this way, we compute $\gamma_{ptb} \approx 12$ bar for the test-case BHPs in Fig. 2.2. The POD-TPWL errors for this test case, computed as described above, are $E_r = 0.0302$, $E_X = 0.0674$, $E_{ps} = 0.0033$, and $E_\sigma = 0.0017$. These values indeed fall within the range of the 20 test cases in Fig. 2.14 for $\pm 10$ bar for ‘3 regular trainings,’ which is the closest set of runs in the systematic assessment. Thus we see that the detailed results presented earlier can be viewed as (at least approximately) representative of test-case results for $\gamma_{ptb} = 10$ bar with ‘3 regular trainings.’

### 2.4 Model 2.2: 3D Example

We now apply our POD-TPWL model for a prototype 3D case. The overall model, shown in Fig. 2.15, contains $90 \times 55 \times 3$ grid blocks (total of 14,850 blocks). The central reservoir region is modeled using $70 \times 35 \times 3$ grid blocks, each of dimensions $10\text{ ft} \times 8\text{ ft} \times 5\text{ ft}$. The full geomechanical domain is of overall physical dimension $900\text{ ft} \times 440\text{ ft} \times 15\text{ ft}$, and is represented using $91 \times 56 \times 4$ nodes (total of 20,384 nodes). In this model, we include a border region of low permeability (dark blue region in Fig. 2.15), extending 10 grid blocks in the $x$ and $y$ directions around the central reservoir. This border region (along with other problem parameters) is important for robust AD-GPRS performance for 3D coupled flow-geomechanics problems, and we have observed performance to be sensitive to the number of layers in the border region.

The log-permeability field of the central reservoir, shown in Fig. 2.15, is Gaussian. The correlation structure in the $x$ and $y$ directions is the same as in Model 2.1. The arithmetic mean permeability is also the same as in Model 2.1. We set $k_x = k_y = k_z$. The porosity of the reservoir is constant at $0.2$. Flow is driven by two injectors and a producer, as indicated in Fig. 2.15. These wells are completed only in layer 2 of the model.

The initial pressure is set to 413 bar at the top of the reservoir. Young’s modulus is determined based on the initial stress field, and is set to 6.9 GPa at the top of the reservoir. Poisson’s ratio is specified to be 0.2. A constant stress of 900 bar is imposed
on the top boundary. No-displacement boundary conditions are applied at the other five boundaries. The simulation time frame is 700 days. The full-order system has a total $14,850 \times 2 + 20,384 \times 3 = 90,852$ primary variables.

![3D model (Model 2.2) setup](image)

Figure 2.15: 3D model (Model 2.2) setup

To construct the POD-TPWL model, we perform three full-order training simulations. The training and test-case BHP schedules for the injection wells are shown in Fig. 2.16. The BHP profile for production well P1 is the same as in Model 2.1, and again does not change between the training and test runs. A total of 105 snapshots are generated in the three training runs, and we set $l_p = l_S = l_{d_x} = l_{d_y} = l_{d_z} = 60$ for the POD-TPWL model.

Figure 2.17 displays POD-TPWL results for the well rates. All curves are as previously defined. We observe close agreement between POD-TPWL and AD-GPRS for this prototype 3D example. The POD-TPWL test-case results are generally, but not always, within the range of the training runs. It is evident that the training solution ‘nearest’ to the test-case solution changes during the course of the run, sometimes rather abruptly (e.g., I1 injection rate, Fig. 2.17(a), at around 400 days). It is encouraging to see that the POD-TPWL model retains accuracy even in these instances.

Maps of saturation, maximum principal stress and minimum principal stress, in layer 2 of the model at 600 days, are shown in Figures 2.18, 2.19 and 2.20. The $y$ axes in all of these figures are stretched by about a factor of 2 for improved clarity. The format for these figures is as described in the context of Model 2.1. We see that
the POD-TPWL and HFS models ((a) and (b)) in each figure) agree very closely visually. The degree of similarity between the closest HFS training run and the test run is displayed in Figures 2.18(c), 2.19(c) and 2.20(c). The impact of piecewise linearization is evident by comparing these figures with Figures 2.18(d), 2.19(d) and 2.20(d). More specifically, for saturation and both principal stresses, the error maps (in (d)) indicate a high degree of POD-TPWL solution accuracy, consistent with the close visual correspondence in the (a) to (b) comparisons.

We present POD-TPWL results for $\chi(t)$ at the injection well locations in Fig. 2.21. For both wells, the POD-TPWL test-case predictions closely track those of AD-GPRS (minor discrepancies are however apparent in Fig. 2.21(b)). We see that, for both injection wells, $\chi$ passes through zero at around 600 days, indicating rock failure at these locations. This behavior is captured accurately by the POD-TPWL model.

We note finally that the AD-GPRS and POD-TPWL timings for this case are each about 10-20% higher than those for the 2D model. Thus we achieve similar runtime speedup in this case. Timings can again vary depending on the state of the computational cluster.
Figure 2.17: Well rates for test case (Model 2.2)
2.4. MODEL 2.2: 3D EXAMPLE

Figure 2.18: Oil saturation in layer 2 at 600 days (Model 2.2)
Figure 2.19: Maximum effective principal stress ($\sigma'_1$) in layer 2 at 600 days (Model 2.2)
2.4. MODEL 2.2: 3D EXAMPLE

Figure 2.20: Minimum effective principal stress ($\sigma_3'$) in layer 2 at 600 days (Model 2.2)
2.5 Summary

In this chapter, we introduced a POD-TPWL reduced-order modeling procedure to treat coupled flow-geomechanics problems. Our implementation required a comprehensive extension of the general POD-TPWL methodology to handle geomechanical equilibrium equations and the associated displacement variables, along with the oil-water flow equations and variables. The formulation included the use of multiple derivatives for linearization. The coupled flow-geomechanics POD-TPWL model was tested extensively on a 2D case and was also applied to a prototype 3D example. Generally accurate POD-TPWL results for key quantities of interest, including well injection and phase production rates, well failure criterion, and global pressure, saturation and principal stress fields, were achieved. Online (runtime) speedups of $O(100)$, relative to optimized AD-GPRS simulations, were observed for the cases considered. For the 2D system, we also studied the impact of the training procedure and the level of perturbation (in test-case BHP profiles relative to those in the training runs) on POD-TPWL error for key quantities of interest.

Figure 2.21: Mohr-Coulomb failure criterion ($\chi$) at the injection wells (Model 2.2)
Chapter 3

POD-TPWL for CO₂ Storage Operations

In this chapter, we refine our earlier POD-TPWL treatment [58] for modeling CO₂ storage problems. We first present the governing equations and then describe the POD-TPWL ROM formulation and enhancements for this problem. Our procedure for linearizing around different training runs (referred to as multiple derivatives) will be discussed in detail. Field-scale results for two 3D systems are then presented. These demonstrate the impact of using the multiple-derivative treatment in terms of error reduction. Finally, we apply POD-TPWL for MADS-based optimization, in which a measure of CO₂ storage risk is minimized. We note that there is some overlap here with [58], since we intend for this chapter to be self-contained.

3.1 POD-TPWL for CO₂-Water System

In this section, we present the POD-TPWL reduced-order model for a CO₂-water system. The formulation presented here generally follows that in Chapter 2, though there are important differences since geomechanical effects are not considered, and the compositional system here differs from the oil-water system in Chapter 2. As noted in Chapter 1, the work in this chapter was actually performed prior to that in Chapter 2, and this contribution represents an extension of the author’s Master’s thesis [58].
The POD-TPWL formulation for CO₂-water systems closely resembles that for oil-gas compositional models presented by He and Durlofsky [46, 47]. New aspects of the formulation here, relative to the developments in [46, 47, 58], include the use of derivatives from multiple training simulations and the application of POD-TPWL for optimization in CO₂ storage operations. Further description of some of the detailed treatments, such as the specification of time-varying rates (rather than bottom-hole pressure) for wells, can be found in [58].

3.1.1 Governing equations

The CO₂-water flow equations resemble Eq. 2.1 in Section 2.1.1. However, instead of the oil-water system described by Eq. 2.1, here we use the compositional formulation in AD-GPRS [126]. With this formulation, the solubility of CO₂ is treated as a function of both pressure and composition in the water phase, which provides a more accurate representation of dissolution trapping of CO₂. The governing equations consist of mass conservation statements for the CO₂ (denoted as CO₂) and water (denoted as w) components, which can exist in either gas or water phases (designated by g and w), combined with Darcy’s law for each phase. Following the presentation by Jin [58], the resulting equation for each component \( c \) (\( c = \text{CO}_2, w \)), with capillary pressure effects neglected, is

\[
\frac{\partial}{\partial t} \left[ \phi (S_w \rho_w x_c + S_g \rho_g y_c) \right] - \nabla \cdot \left\{ k \left[ \lambda_w \rho_w x_c (\nabla p + \rho_w g \nabla D) + \lambda_g \rho_g y_c (\nabla p + \rho_g g \nabla D) \right] \right\} + \sum_{k=1}^{n_w} (\rho_w x_c q_k^w + \rho_g y_c q_k^g) = 0,
\]

where the subscript \( j = g, w \) denotes fluid phase. The geological characterization is represented in Eq. 3.1 through porosity \( \phi \) and permeability tensor \( k \), while the interaction between the rock and fluids is specified by phase mobility \( \lambda_j \), where \( \lambda_j = k_{rj}/\mu_j \), with \( k_{rj} \) denoting the relative permeability of phase \( j \), and \( \mu_j \) is the viscosity of phase \( j \). The solution variables considered here are pressure \( p \) and \( z_c \) (overall molar fraction of component \( c \)), given by \( z_c = S_g y_c + S_w x_c \), where \( S_g \) and \( S_w \) represent gas and water phase saturation, and \( x_c \) and \( y_c \) indicate the molar fractions of component \( c \) in the water and gas phases. Other variables include time \( t \), phase density \( \rho_j \),
3.1. **POD-TPWL FOR CO$_2$-WATER SYSTEM**

The compositional system described in Eq. 3.1 is completed by enforcing the saturation constraint $S_g + S_w = 1$. Following [46, 58], the molar formulation is applied to solve Eq. 3.1. In this case, the two primary variables, which fully define the system, are pressure $p$ (same for both phases) and the overall molar fraction of the water component $z_w$. Note that $z_{CO_2} = 1 - z_w$, where $z_{CO_2}$ is the overall molar fraction of the CO$_2$ component. As discussed by Voskov and Tchelepi [110], other choices can also be used for the primary variables. As explained by Voskov and Tchelepi [111], the full system has a total of eight unknowns in the general case. Please consult [46, 58] for a detailed description of the compositional equations.

Since we consider only flow (rather than both flow and geomechanics, as in Chapter 2), which is fully defined by two primary variables in each grid block, the total number of variables in the system is now $n_v = 2n_b$. Here $n_b$ is again the total number of grid blocks in the model. The set of nonlinear algebraic equations representing the system, after discretization using a standard finite volume formulation with a fully implicit time treatment, has the same form as Eq. 2.8 (repeated here for completeness):

$$g(x^{n+1}, x^n, u^{n+1}) = 0,$$

where $g \in \mathbb{R}^{2n_b}$ is the residual vector we seek to drive to zero, and $x \in \mathbb{R}^{2n_b}$ represents the system states; i.e., $x = [(p, z_w)_1, \ldots, (p, z_w)_{n_b}]^T$. Again, the superscripts $n$ and $n+1$ designate time level, and $u^{n+1}$ represents the (specified) well control parameters. The only unknown is $x^{n+1}$.

In the formulation in Chapter 2, and in most of the previous POD-TPWL implementations (including [46]), wells were controlled by specifying wellbore pressure at a particular location, which is referred to as bottom-hole pressure (BHP). For the CO$_2$ storage problems discussed in this chapter, since we typically wish to inject a specified volume of CO$_2$ at each time step, following [58], we operate the wells under rate control. The modification of the AD-GPRS Jacobian matrix output for rate-controlled problems was described in [58], which should be consulted for details.

As discussed in Section 2.1.1, the nonlinear system defined by Eq. 3.2 is typically
solved using Newton’s method. In this case, the Jacobian matrix is of dimensions $2n_b \times 2n_b$. This linear solution process is typically the most time consuming part of the flow simulation, especially for models involving relatively few components (as is the case here). POD-TPWL is again appealing since it avoids the construction and solution of this $2n_b \times 2n_b$ system during online computations.

### 3.1.2 Linearized representation

The POD-TPWL development in this chapter is analogous to that for the coupled flow-geomechanics systems presented in Chapter 2. However, because different variables are considered here, and because the relevant vectors and matrices are of different dimensions, we repeat some of the development and POD-TPWL equations here. As discussed in Chapter 2, POD-TPWL involves linearization around training runs, representation of solution variables using POD, and projection (or constraint reduction). We now describe each of these steps, starting with linearization.

We again use the superscripts $i$ and $i + 1$ to designate time steps in the training runs, and $n$ and $n + 1$ to represent time steps in test (new) runs. The residual vector for the test simulation at time step $n + 1$ is given by

$$g^{n+1} = 0 \approx g^{i+1} + \frac{\partial g^{i+1}}{\partial x^{i+1}}(x^{n+1} - x^{i+1}) + \frac{\partial g^{i+1}}{\partial x^i}(x^n - x^i) + \frac{\partial g^{i+1}}{\partial u^{i+1}}(u^{n+1} - u^{i+1}), \quad (3.3)$$

where $g^{i+1} = g(x^{i+1}, x^i, u^{i+1}) = 0$, and $g^{n+1} = g(x^{n+1}, x^n, u^{n+1}) = 0$, as noted in Chapter 2. After rearrangement, Eq. 3.3 can be written as:

$$J^{i+1}(x^{n+1} - x^{i+1}) = - [A^{i+1}(x^n - x^i) + B^{i+1}(u^{n+1} - u^{i+1})], \quad (3.4)$$

with the three matrices defined as

$$J^{i+1} = \frac{\partial g^{i+1}}{\partial x^{i+1}} \in \mathbb{R}^{2n_b \times 2n_b}, \quad A^{i+1} = \frac{\partial g^{i+1}}{\partial x^i} \in \mathbb{R}^{2n_b \times 2n_b}, \quad B^{i+1} = \frac{\partial g^{i+1}}{\partial u^{i+1}} \in \mathbb{R}^{2n_b \times n_u}. \quad (3.5)$$

Here $J^{i+1}$ is again the Jacobian matrix at time step $i + 1$ of the training simulation, evaluated for the converged system. We note that Eqs. 3.3 - 3.5 here are analogous to Eqs. 2.9 - 2.11 in Section 2.1.2.
3.1.3 POD and constraint reduction

As discussed in Chapter 2, we apply proper orthogonal decomposition (POD) to generate a basis matrix \( \Phi \in \mathbb{R}^{2n_b \times l} \). We then use this basis matrix to represent the state variables \( \mathbf{x} \in \mathbb{R}^{2n_b} \) in terms of a reduced state vector \( \xi \in \mathbb{R}^l \) (where \( l \) is the dimension of the subspace); i.e.,

\[
\mathbf{x} \approx \Phi \xi. \tag{3.6}
\]

The basis matrix \( \Phi \) here has only the flow component (rather than both flow and geomechanics, as in Section 2.1.3). The basis matrix \( \Phi \) is constructed by performing singular value decomposition (SVD) of the snapshot matrices, generated from training run solutions for the two state variables \( p \) and \( z_w \). We thus form the snapshot matrices \( \mathbf{X}_p \in \mathbb{R}^{n_b \times L} \) and \( \mathbf{X}_z \in \mathbb{R}^{n_b \times L} \), where \( L \) is the total number of snapshots from all training simulations, as

\[
\mathbf{X}_p = [p_1, p_2, \ldots, p_L], \quad \mathbf{X}_z = [z_{w1}, z_{w2}, \ldots, z_{wL}]. \tag{3.7}
\]

Here the vectors \( p^i \in \mathbb{R}^{n_b} \) and \( z_{wi}^i \in \mathbb{R}^{n_b}, i = 1, \ldots, L \), indicate the pressure and overall molar fraction of the water component, in each of the \( n_b \) grid blocks, at a particular time step in a training run.

The SVD operations are performed separately for \( \mathbf{X}_p \) and \( \mathbf{X}_z \). A limited number of left singular vectors are used (as columns) to construct the basis matrices \( \Phi_p \) and \( \Phi_z \). The exact number of vectors to retain, i.e., \( l_p \) columns in \( \Phi_p \) and \( l_z \) columns in \( \Phi_z \) (\( l_p \) and \( l_z \) can differ), are determined based on an energy criterion, or limited numerical experimentation [46, 47], as discussed in Section 2.1.3.

Given \( \Phi_p \) and \( \Phi_z \), the state variables are represented as:

\[
\mathbf{x} = \begin{bmatrix} \mathbf{x}_p \\ \mathbf{x}_z \end{bmatrix} \approx \Phi \xi = \begin{bmatrix} \Phi_p & 0 \\ 0 & \Phi_z \end{bmatrix} \begin{bmatrix} \xi_p \\ \xi_z \end{bmatrix}, \tag{3.8}
\]

where \( \xi_p \in \mathbb{R}^{l_p} \) and \( \xi_z \in \mathbb{R}^{l_z} \) are the reduced variables for pressure and overall molar fraction of the water component. This projection reduces the number of primary unknowns from \( 2n_b \) to \( l \), where \( l = l_p + l_z \), with \( l \ll 2n_b \). The structure of the overall \( \Phi \) matrix shown in Eq. 3.8 only applies if we order the unknowns with pressure
in all blocks appearing first, followed by molar fraction of water in all blocks (i.e., \( p_1, p_2, \ldots, p_{n_b}, z_{w,1}, z_{w,2}, \ldots, z_{w,n_b} \)). Consistent with the discussion in Section 2.1.4, the unknowns are ordered as \((p_1, z_{w,1}), (p_2, z_{w,2}), \ldots, (p_{n_b}, z_{w,n_b})\) in practice to maintain compatibility with the Jacobian matrices generated by AD-GPRS. Therefore, the entries in \( \Phi_p \) and \( \Phi_z \) are interspersed within the overall \( \Phi \) matrix.

Following the development in Section 2.1.3, we insert the representation for the states in Eq. 3.8 into (linearized) Eq. 3.4, which gives a (over-determined) system of \( 2n_b \) equations in \( l \) variables. To render the system well posed, we premultiply the resulting set of equations by \( \Psi^T \), where the matrix \( \Psi \in \mathbb{R}^{2n_b \times l} \) is again the constraint reduction matrix. Consistent with the discussion in Section 2.1.3, we apply least-squares Petrov Galerkin (LSPG) projection for constraint reduction. This means we set \( \psi^{i+1} = J^{i+1} \Phi \). The low-order linearized equation is now given by:

\[
\Psi^T J^{i+1} \Phi (\xi^{n+1} - \xi^{i+1}) = -\Psi^T [A^{i+1} \Phi (\xi^n - \xi^i) + B^{i+1} (u^{n+1} - u^{i+1})],
\]

which involves \( l \) equations and \( l \) unknowns. This equation can be expressed as

\[
\xi^{n+1} = \xi^{i+1} - (J^{i+1}_r)^{-1} [A^{i+1}_r (\xi^n - \xi^i) + B^{i+1}_r (u^{n+1} - u^{i+1})],
\]

where the reduced derivative matrices are defined as

\[
J^{i+1}_r = (\Psi^{i+1})^T J^{i+1} \Phi, \quad A^{i+1}_r = (\Psi^{i+1})^T A^{i+1} \Phi, \quad B^{i+1}_r = (\Psi^{i+1})^T B^{i+1}.
\]

Here, \( J^{i+1}_r \in \mathbb{R}^{l \times l} \), \( A^{i+1}_r \in \mathbb{R}^{l \times l} \), and \( B^{i+1}_r \in \mathbb{R}^{l \times nu} \). Eqs. 3.9 - 3.11 here are analogous to Eqs. 2.18 - 2.20 in Section 2.1.3.

### 3.1.4 Multiple-derivative point selection

The point selection procedure refers to the process applied at each (online) time step in a POD-TPWL run, in which we determine the sequential training states \( i \) and \( i+1 \) to linearize around (see Eqs. 3.10 and 3.11). In other words, during this procedure, we select the training ‘point’ around which the linearization is performed. In most previous work with POD-TPWL (including [58]), the points used for linearization all derived from one training run (referred to as the primary training run). This type
of approach will be called the single-derivative method. Here we apply a multiple-derivative method in which we use derivative information from more than one training run. As we will see, this can lead to significant improvement in POD-TPWL accuracy.

Figure 3.1 illustrates the multiple-derivative treatment. For illustration purposes this figure depicts point selection in high-dimensional (state-variable) space, though it is actually performed in reduced space, as explained below. The gray line in Fig. 3.1 represents the trajectory for one training run (for which we specify controls $u^0$), the blue line is the trajectory for the test simulation ($u^2$), and the red line is the trajectory for an additional training run ($u^1$). The $x$ and $y$ axes represent the CO$_2$ molar fraction and pressure. As we can observe from the figure, when $n = 1, \cdots, 4$, the states in training run $u^0$ are closer to the test case $u^2$, so POD-TPWL would entail linearization around the appropriate point in training run $u^0$. However, for $n = 5, 6$, the states in the test run are closer to the second training run $u^1$, so the derivatives from that case would now be used in the POD-TPWL representation. In most previous formulations, either one or the other (but not both) of these training runs would be used in POD-TPWL.

During a test run, at each time step $n$, the linearization point in the training set is found by minimizing a measure of distance between the current (reduced) test-run state and the points in the training runs. In order to achieve this, we need to determine (1) the training run $\tau$ ($\tau = 1, 2, \cdots, n_{\text{train}}$, where $n_{\text{train}}$ is the total number of

Figure 3.1: POD-TPWL with derivatives from two training runs
training runs), and (2) the step \( j \) in training run \( \tau \). Following He and Durlofsky [46] and Jin [58], we define this distance \( d = d(n, j, \tau) \) as

\[
d = d_z + \gamma d_{\text{cwi}}. \tag{3.12}
\]

Here \( d_z \) and \( d_{\text{cwi}} \) represent the relative difference in reduced molar fraction and in estimated cumulative well injection, and \( \gamma \) is a weighting parameter (we take \( \gamma = 3 \) based on limited numerical experimentation). The definitions of \( d_z \) and \( d_{\text{cwi}} \) are

\[
d_z = d_z(n, j, \tau) = \frac{|\xi_z^n - \xi_z^{\tau_j}|}{|\xi_z^n| + \epsilon}, \tag{3.13}
\]

and

\[
d_{\text{cwi}} = d_{\text{cwi}}(n, j, \tau) = \sum_{k=1}^{n_w} \frac{\left|\int_0^{t^n} q_k^n dt - \int_0^{t^n} q_k^{\tau_j} dt\right|}{\int_0^{t^n} q_k^n dt + \epsilon}, \tag{3.14}
\]

where in Eq. 3.13, the variables \( \xi_z^n \) and \( \xi_z^{\tau_j} \) designate reduced overall molar fraction of water for time step \( n \) in the test run and for the saved point \( j \) in training run \( \tau \), respectively. The variables \( q_k^n \) and \( q_k^{\tau_j} \) in Eq. 3.14 denote the injection rate for well \( k \) at time step \( n \) in the test run and at time step \( j \) in training run \( \tau \). In both definitions, the \( \epsilon \) term (set to 0.01) is included to avoid dividing by very small values, which is relevant only at early times. Note that the definition of \( d_{\text{cwi}} \) involves the relative error in the cumulative injection for each well individually, instead of only considering the entire field as a whole. The point selection procedure finds the point in the training runs that is closest to the current solution (in terms of \( \xi_z \) and cumulative well injection) by finding the values of \( j^* \) and \( \tau^* \) that minimize \( d \) in Eq. 3.12.

Here \( j^* \) and \( \tau^* \) are determined simultaneously as follows. For step \( n \) in the test run, a set of possible linearization points is identified in each training run \( \tau \). This set of points is denoted as \( D^n_\tau \), with \( D^n_\tau = \{n - 10, \cdots , n - 1, n, n + 1, \cdots , n + 10\} \); i.e., we consider 21 points in each training run. The optimal linearization point (\( j^* \) and \( \tau^* \)) for step \( n \) is then selected as follows

\[
(j^*, \tau^*)^n = \arg\min_{j \in D^n_\tau, \tau \in \{1, \cdots , n_{\text{train}}\}} d(n, j, \tau). \tag{3.15}
\]
The operation count for point selection is $O(l)$, though the coefficient can be large since a considerable number of potential points are considered. As a result, using derivatives from five training runs, point selection with our current implementation can consume up to 40% of the online runtime. For this and other reasons, we do not advocate using large numbers of training runs for linearization (i.e., we suggest using $n_{\text{train}} \lesssim 10$).

### 3.1.5 POD-TPWL workflow

We now provide a brief review of the steps entailed in generating and running the POD-TPWL model for CO$_2$ storage simulations. The workflow for the overall procedure is presented in Algorithm 3. Note that Algorithm 3 closely resembles Algorithms 1 and 2 in Chapter 2, though here flash calculations are required. Starting with the preprocessing (offline) computations, we perform training simulations, generate basis matrices $\Phi_p$ and $\Phi_z$, and then construct the reduced derivative matrices appearing in Eqs. 3.10 and 3.11. AD-GPRS was previously modified to provide the necessary derivative information required for the rate-controlled setup. Additional computations, corresponding to about one-third of the total time required for all of the training runs, are required in the offline step. In this work we use five training runs unless otherwise stated, in which case the total overhead (training runs plus construction of reduced matrices) corresponds to about 6.7 full-order simulations. These requirements are for serial computation – less overhead, in terms of elapsed time, would be entailed if the preprocessing computations were performed in parallel.

For the CO$_2$ storage simulations considered here, runtime speedups using POD-TPWL of over a factor of 100 are observed, and even larger speedups may be expected for larger and more complex models [46]. Therefore, the POD-TPWL procedure is considered very cost effective if the model is to be run a large number of times (as is the case for optimization). If, however, the task only requires a few simulation runs, it is more efficient to simply run the full-order model.

During the online POD-TPWL computations, the training states $i$ and $i + 1$ are found by minimizing $d(n, j, \tau)$, given by Eq. 3.12, at each time step. Eq. 3.10 is subsequently solved to determine $\xi^{n+1}$. If required, we reconstruct the actual states by applying $x^{n+1} \approx \Phi \xi^{n+1}$. In many applications, states are only required at well
blocks, where they are used to compute production and injection quantities (a flash must be performed to first construct secondary variables such as saturation). In some of our computations we are interested in the distribution of state variables; e.g., molar fraction of CO$_2$ at a target layer, rather than just well quantities, so states must also be constructed at particular locations within the domain.

Algorithm 3 POD-TPWL procedures

1: procedure Offline Processing
2: Perform training runs with given control settings;
3: Load snapshots $X_p, X_z$ (Eq. 3.7) and derivatives $J^i, A^i, U^i$ from each training;
4: Construct basis matrix $\Phi$ (Eq. 3.8) from snapshots of training runs;
5: Construct reduced states $\xi^i$ (using $\xi^i = \Phi^T x^i$) and reduced derivatives $J^i_r, A^i_r, B^i_r$ (Eq. 3.11) for all trainings;
6: end procedure

7: procedure Online Processing
8: for $n = 1$ to $T$ (final simulation time) do
9: Find training state $\xi^i_{k^*}$ that is closest to $\xi^n$ (using Eq. 3.12);
10: Solve Eq. 3.10 for $\xi^{n+1}$;
11: end for
12: Reconstruct primary variables $x^{n+1}$ from $\xi^{n+1}$ at required locations (well blocks, target layers) using Eq. 3.8;
13: Perform flash calculations with primary variables to construct secondary variables;
14: Compute well responses and target functions.
15: end procedure

3.2 CO$_2$ Storage Simulation using POD-TPWL

In this section, we present results using POD-TPWL for the simulation of the injection phase of CO$_2$ storage operations. The method will be tested on two field-scale cases, one involving a Gaussian log-permeability field, and the other a fluvial channelized system. The advantages of using multiple derivatives (from multiple training runs) will be assessed in both cases. The use of POD-TPWL models in optimization will be demonstrated in Section 3.3.
3.2. 

The basic model used here is similar to that considered by Cameron and Durlofsky [12]. Dissolution of CO$_2$ is incorporated within the compositional treatment. As noted earlier, capillary pressure is not included, as we expect its impact to be small given the large grid blocks used in the model (these effects are important in the equilibration stage, which is not modeled here). The gas-water relative permeability curves, taken from Saadatpoor [98], are shown in Fig. 3.2. Relative permeability hysteresis is not included. This simplification is probably reasonable during the injection period, though hysteresis is important during equilibration [64]. Finally, chemical reactions and mineralization are neglected, which is again appropriate during the injection stage.

3.2.1 Model 3.1: Gaussian log-permeability field

The simulation model, shown in Fig. 3.3(a), includes a storage aquifer, of physical dimensions 2.8 km $\times$ 2.8 km $\times$ 150 m, immersed within a large-scale regional model, of dimensions 43.4 km $\times$ 43.4 km $\times$ 150 m. The storage aquifer is modeled on a 35 $\times$ 35 $\times$ 15 grid (total of 18,375 grid blocks), while the full system is represented on a 49 $\times$ 49 $\times$ 15 grid (total of 36,015 grid blocks). In the storage aquifer, grid blocks
are of size 80 m \times 80 m \times 10 m. Block sizes increase as we move out from the storage aquifer, as is evident in Fig. 3.3(c). Note that the grid is still fairly dense just outside the storage aquifer. We use a Gaussian log-permeability field with a mean of 1 and a variance of 1. For this case, we take \( k_x = k_y = k_z \) over most of the domain. In layer 7, which is a low-permeability partial-confinement layer, we specify \( k_z = 0.1 k_x \) (this layer is evident in Fig. 3.3(a)). The porosity of the aquifer is set to a constant of 0.2. \( \text{CO}_2 \) injection is accomplished using four horizontal wells, and their locations are shown in Fig. 3.3(b).

![Image](image.png)

(a) \( \log k_z \) for storage aquifer (b) Horizontal well locations (c) Full model (46 \times 46)

Figure 3.3: Model 3.1 setup (Gaussian log-permeability field)

The simulation model contains supercritical \( \text{CO}_2 \) and water components in gas and water phases. The initial pressure is 170 bar at the top of the reservoir. The four horizontal wells are located in layer 13 (third layer from the bottom). The lengths of these wells range from 400 m to 640 m. We control the wells by specifying injection rates. The model is run for a total of 7300 days (about 20 years). The full-order system contains 36,015 \times 2 = 72,030 primary variables.

Five full-order training simulations, with time-varying injection rates, are performed to generate state and derivative information for the POD-TPWL model. The rate profiles (at injection-well conditions of 372 K, \(~190 \text{ bar}) of the four wells for each of these training runs are shown in Fig. 3.4. Injection rates are varied every 4 years. These training profiles, as well as the test profiles considered below, correspond to profiles observed during the course of a computational optimization procedure (in general, the controls used in training runs should correspond as closely as possible to those that will be applied during online computations). The mass injection rate
from all four wells in training and test runs is approximately 0.22 million tonnes of CO$_2$ per year (overall volumetric injection rate and CO$_2$ density, at well conditions, are about 1290 m$^3$/day and 459 kg/m$^3$, respectively). The cumulative CO$_2$ injected after 7300 days is about 4% of the pore volume of the storage aquifer. A total of 600 snapshots were generated in the five full-order training runs, and we use $l_p = l_z = 80$ in the POD-TPWL model. In general, decreasing $l_p$ and $l_z$ will accelerate the POD-TPWL runs, though accuracy may be reduced. Increasing $l_p$ and $l_z$ will enhance accuracy up to a point, though eventually over-fitting may occur, which can lead to a decrease in accuracy in test runs. The use of $l_p = l_z = 80$ was determined based on a few numerical experiments. These values are in the general range of those used by He and Durlofsky [47].

We now assess the performance of POD-TPWL for several test cases. The rate control profiles for these cases are shown in Fig. 3.5. As in the training runs, injection rates are varied every 4 years during the 20-year injection period. Because injection rates are specified, the well quantity of interest is the well bottom-hole pressure (BHP).

Figures 3.6 and 3.7 display results for injection-well BHPs for Test Case 1. The gray curves denote results for training cases, the red curves define the full-order reference AD-GPRS solution, and the blue curves display the POD-TPWL results. The plots on the left in Figs. 3.6 and 3.7 correspond to POD-TPWL results using derivatives from only a single (primary) training run. The gray curve in each of those plots displays the results for the training run used to provide the derivative information. Note that, in these single-derivative results, snapshots from all five training runs are still used to construct the basis matrix. The plots on the right in Figs. 3.6 and 3.7 correspond to POD-TPWL results using derivatives from all five trainings (most of the multiple-derivative results presented in this chapter use derivatives from five training runs, though later we will show results with two, three and four trainings). Agreement between the POD-TPWL model and the full-order reference results is clearly improved when multiple training runs are used. This is most evident for Wells 2 and 3. We note finally that the primary training run and Test Case 1 involve significantly different well controls, which is why the single-derivative treatment displays clear errors. In cases where the training and test controls are
Figure 3.4: Injection schedules at reservoir conditions for training runs (Model 3.1)
Figure 3.5: Injection schedules for test cases (Model 3.1)
‘closer,’ errors using this approach will be smaller.

Figure 3.6: BHP responses for Test Case 1, Wells 1 and 2 (Model 3.1)

In addition to well quantities, we are also interested in the location of the injected CO$_2$ within the aquifer. This type of global field is usually not the focus in well control optimization problems, and accuracy in such quantities can be challenging for ROM procedures. Fig. 3.8(a) displays a map of CO$_2$ overall molar fraction at the end of the injection period (20 years) in layer 6, which is just above the partial-confinement layer 7, in the primary training case. Fig. 3.8(b) presents $z_{CO_2}$ for the full-order test simulation. Maps displaying the difference between POD-TPWL and full-order results for $z_{CO_2}$ in Test Case 1 are shown in Figs. 3.8(c) and (d). Although
3.2. CO$_2$ STORAGE SIMULATION USING POD-TPWL

Figure 3.7: BHP responses for Test Case 1, Wells 3 and 4 (Model 3.1)
CHAPTER 3. POD-TPWL FOR CO$_2$ STORAGE OPERATIONS

Figure 3.8: Overall molar fraction of CO$_2$ at the end of injection in layer 6 for Test Case 1 (Model 3.1)
the color bar spans a much smaller range here than in Figs. 3.8(a) and (b), it is evident that the multiple-derivative treatment results in significantly smaller errors than the single-derivative treatment.

![Figure 3.9: Aggregate CO\(_2\) in layer 6 for Test Case 1 (Model 3.1)](image)

To quantify the total CO\(_2\) in a target layer (layer 6 in this case) at a particular time, it is useful to introduce the aggregate quantity \(Z_{CO_2}(t)\), which we define as

\[
Z_{CO_2}(t) = \sum_{i=1}^{n_{il}} z_{CO_2,i}(t),
\]

where \(z_{CO_2,i}\) represents the CO\(_2\) molar fraction in cell \(i\), and \(n_{il}\) denotes the total number of grid blocks in the target layer. Fig. 3.9 displays the time evolution of aggregate CO\(_2\) for POD-TPWL runs using both single and multiple derivatives. Consistent with previous observations, we see that the multiple-derivative treatment leads to significantly better accuracy, relative to reference AD-GPRS results, for this important quantity.

We now quantify the error in time-varying BHP and overall CO\(_2\) molar fraction for five different test cases (including Test Case 1 considered above). These error measures are defined as

\[
\epsilon_{BHP} = \frac{1}{n_w} \sum_{i=1}^{n_w} \frac{1}{T} \int_0^T \left| p_{i,\text{TPWL}} - p_{i,\text{HFS}} \right| dt,
\]

where
\[
\epsilon_{CO_2} = \frac{1}{T} \int_0^T \sum_{i=1}^{n_w} \left| z_{CO_2,i}^{TPWL} - z_{CO_2,i}^{HFS} \right| \, dt,
\]  
(3.18)

where \( n_w \) denotes the number of wells in the model, \( T \) is the total simulation time (20 years), \( p_i^{TPWL} \) and \( p_i^{HFS} \) represent the well BHPs for the reduced-order and high-fidelity simulations for well \( i \), and \( z_{CO_2,i}^{TPWL} \) and \( z_{CO_2,i}^{HFS} \) represent the CO\(_2\) molar fractions in cell \( i \). Note that here we use absolute rather than relative errors (as were used in Section 2.3). We compute \( \epsilon_{BHP} \) and \( \epsilon_{CO_2} \) for single-derivative and multiple-derivative treatments with two, three, four and five derivatives. These results, therefore, enable us to gauge the impact of using varying numbers of derivatives.

Results for \( \epsilon_{BHP} \) and \( \epsilon_{CO_2} \) are presented in Tables 3.1 and 3.2. The last row in each table provides the average over the five test cases. The average errors for both quantities decrease monotonically with increasing numbers of derivatives, as would generally be expected. For a given test case, however, this decrease is not always strictly monotonic. This suggests that the point selection procedure, although quite effective, could be further refined. In any event, as we proceed from the use of the single-derivative treatment to the use of all five derivatives, the average \( \epsilon_{BHP} \) and \( \epsilon_{CO_2} \) decrease by about 63% and 81%, respectively. This demonstrates the efficacy of the multiple-derivative treatment.

Table 3.1: POD-TPWL BHP errors (in bar) for five test cases using varying numbers of derivatives (Model 3.1)

<table>
<thead>
<tr>
<th></th>
<th>1 derivative</th>
<th>2 derivatives</th>
<th>3 derivatives</th>
<th>4 derivatives</th>
<th>5 derivatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case 1</td>
<td>0.2839</td>
<td>0.0702</td>
<td>0.0702</td>
<td>0.0702</td>
<td>0.0702</td>
</tr>
<tr>
<td>Test Case 2</td>
<td>0.3357</td>
<td>0.2767</td>
<td>0.2767</td>
<td>0.2767</td>
<td>0.1495</td>
</tr>
<tr>
<td>Test Case 3</td>
<td>0.3453</td>
<td>0.2871</td>
<td>0.2871</td>
<td>0.1846</td>
<td>0.1846</td>
</tr>
<tr>
<td>Test Case 4</td>
<td>0.5243</td>
<td>0.5243</td>
<td>0.1879</td>
<td>0.1833</td>
<td>0.1926</td>
</tr>
<tr>
<td>Test Case 5</td>
<td>0.2731</td>
<td>0.2626</td>
<td>0.2626</td>
<td>0.0561</td>
<td>0.0561</td>
</tr>
<tr>
<td>Average</td>
<td>0.3525</td>
<td>0.2842</td>
<td>0.2169</td>
<td>0.1541</td>
<td>0.1306</td>
</tr>
</tbody>
</table>

For these test cases, the average time required for the full-order (AD-GPRS) simulations is about 950 seconds per run. Consistent with the overhead requirements given earlier, the serial time required to construct the POD-TPWL model (with five training runs) is about 6300 seconds. The online POD-TPWL runs require about 6.3 seconds per run, which corresponds to a runtime speedup factor of about 150.
3.2. \textit{CO}_2 \textit{STORAGE SIMULATION USING POD-TPWL}

Table 3.2: POD-TPWL $z_{\text{CO}_2}$ errors for five test cases using varying numbers of derivatives (Model 3.1)

<table>
<thead>
<tr>
<th></th>
<th>1 derivative</th>
<th>2 derivatives</th>
<th>3 derivatives</th>
<th>4 derivatives</th>
<th>5 derivatives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case 1</td>
<td>2.6337</td>
<td>0.1335</td>
<td>0.1335</td>
<td>0.1335</td>
<td>0.1335</td>
</tr>
<tr>
<td>Test Case 2</td>
<td>2.7456</td>
<td>1.2718</td>
<td>1.2718</td>
<td>1.2718</td>
<td>0.5309</td>
</tr>
<tr>
<td>Test Case 3</td>
<td>2.7853</td>
<td>1.2555</td>
<td>1.2555</td>
<td>1.2555</td>
<td>0.6952</td>
</tr>
<tr>
<td>Test Case 4</td>
<td>3.0287</td>
<td>3.0287</td>
<td>1.0996</td>
<td>0.8791</td>
<td>1.2066</td>
</tr>
<tr>
<td>Test Case 5</td>
<td>2.5950</td>
<td>1.2974</td>
<td>1.2974</td>
<td>0.1005</td>
<td>0.1005</td>
</tr>
<tr>
<td>Average</td>
<td>2.7577</td>
<td>1.3974</td>
<td>1.0115</td>
<td>0.7280</td>
<td>0.5333</td>
</tr>
</tbody>
</table>

3.2.2 Model 3.2: aquifer with fluvial channels

We now consider injection into a storage aquifer with high-permeability fluvial channels. The quantity of \textit{CO}_2 sequestrated in this case is comparable to that in some existing large-scale projects such as Snøhvit, Sleipner, and In Salah.

Fig. 3.10 displays the channelized permeability field in the storage aquifer, the four horizontal injection wells (of varying length), and the grid used in this case. The storage aquifer (central white square in Fig. 3.10(c)) is of physical dimensions 10.9 km $\times$ 10.9 km $\times$ 100 m. The storage aquifer is immersed within a large-scale regional aquifer, of dimensions 232 km $\times$ 232 km $\times$ 100 m. The full system is represented on a 39 $\times$ 39 $\times$ 10 grid (total of 15,210 grid blocks), with the storage aquifer comprising the central 25 $\times$ 25 $\times$ 10 portion of the model. The grid blocks are of size 436 m $\times$ 436 m $\times$ 10 m in the storage aquifer. For this case, we take $k_x = k_y$ and $k_z = 0.1k_x$. The porosity of the aquifer is again set to 0.2.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure3.10}
\caption{Model 3.2 setup (channelized permeability field)}
\end{figure}
The fluid system considered here is the same as in the first example. The initial aquifer pressure at the top of the model is about 170 bar. The four horizontal wells, located in layer 8 of the model, are of lengths ranging from 872 m to 1308 m. The model is again simulated for a total of 7300 days (about 20 years). The full-order system in this case contains $15,210 \times 2 = 30,420$ primary variables.

To generate state and derivative information for the POD-TPWL model, we again perform five full-order training simulations with time-varying injection rates. Fig. 3.11 shows the rate profiles for the four injection wells for these training cases (at reservoir conditions). The total (field) injection rate in this case is approximately 1.5 million tonnes of CO$_2$ per year. The cumulative CO$_2$ injected after 20 years is about 2.8% of the pore volume of the storage aquifer. A total of 515 snapshots were generated in the five training runs, and we use $l_p = l_z = 80$ for the POD-TPWL model.

The injection schedules for Test Cases 6–10 are shown in Fig. 3.12. The injection rates for these runs are modified every 5 years, consistent with the training runs. As was the case with Model 3.1, the rate profiles for both the training and test runs are consistent with those from an optimization.

The BHP responses of the four wells, for Test Case 6, are shown in Figs. 3.13 and 3.14. The curves are as defined previously, and the plots on the left again correspond to results using derivatives from a single primary training run (with snapshots from all five training runs), and the plots on the right to results using derivatives from all five training runs. We see clear improvement in the results for Wells 1 and 2 (Fig. 3.13) through use of multiple derivatives, but the results for Wells 3 and 4 (Fig. 3.14) are essentially identical with the two treatments.

We now present results for the distribution of CO$_2$ at the target layer (layer 3 in this case). Fig. 3.15 shows the training and test-case simulation results for $z_{CO_2}$ at the end of the injection period, along with difference maps for the test case. The multiple-derivative approach is again seen to provide significantly better accuracy than the single-derivative treatment. Note that the errors using the single-derivative approach are relatively large in this case.

The time evolution of aggregate CO$_2$ ($Z_{CO_2}$ defined in Eq. 3.16) at layer 3 is shown in Fig. 3.16. Again, only the primary training is shown for the single-derivative case on the left. The significant improvement in accuracy in aggregate CO$_2$ through use
3.2. CO$_2$ STORAGE SIMULATION USING POD-TPWL

Figure 3.11: Injection schedules at reservoir conditions for training runs (Model 3.2)
Figure 3.12: Injection schedules for test cases (Model 3.2)
3.2. \textit{CO}_2 \textit{STORAGE SIMULATION USING POD-TPWL}

Figure 3.13: BHP responses for Test Case 6, Wells 1 and 2 (Model 3.2)
Figure 3.14: BHP responses for Test Case 6, Wells 3 and 4 (Model 3.2)
of multiple derivatives is immediately apparent. These results are consistent with the error reduction observed in Fig. 3.15(d) relative to that in Fig. 3.15(c).

The errors (relative to HFS results) in time-varying BHP and overall CO$_2$ molar fraction in layer 3 at the end of the simulation, for Test Cases 6–10, are shown in Table 3.3. We again observe substantial error reduction through use of the multiple-derivative treatment (note that here we only consider the use of derivatives from one and five trainings). More specifically, the average $\epsilon_{\text{BHP}}$ and $\epsilon_{z\text{CO}_2}$ are both reduced by about 77% through use of multiple (five) derivatives.
CHAPTER 3. POD-TPWL FOR CO$_2$ STORAGE OPERATIONS

Figure 3.16: Aggregate CO$_2$ in layer 3 for Test Case 6 (Model 3.2)

Table 3.3: POD-TPWL errors for five test cases (Model 3.2)

<table>
<thead>
<tr>
<th>Test Case</th>
<th>$\epsilon_{\text{BHP}}$</th>
<th>$\epsilon_{\text{CO}_2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 derivative</td>
<td>5 derivatives</td>
</tr>
<tr>
<td>Test Case 6</td>
<td>0.7209</td>
<td>0.2606</td>
</tr>
<tr>
<td>Test Case 7</td>
<td>0.6528</td>
<td>0.0519</td>
</tr>
<tr>
<td>Test Case 8</td>
<td>0.6780</td>
<td>0.0306</td>
</tr>
<tr>
<td>Test Case 9</td>
<td>0.5974</td>
<td>0.2470</td>
</tr>
<tr>
<td>Test Case 10</td>
<td>1.0223</td>
<td>0.2496</td>
</tr>
<tr>
<td>Average</td>
<td>0.7343</td>
<td>0.1679</td>
</tr>
</tbody>
</table>

For this case, the time required to simulate the full-order and POD-TPWL models is 256 seconds and 2.2 seconds, respectively, corresponding to a speedup factor of about 116. The relative times for offline (overhead) computations are analogous to those reported earlier for Model 3.1. Runtime speedup may be less in this case since the full-order models run much faster, and there is still a fixed amount of setup and overhead associated with the POD-TPWL simulations. In general, we expect more substantial speedups for larger and more complicated high-fidelity models.
3.3 Optimization of CO\textsubscript{2} Storage Operations

We now apply the POD-TPWL technique for the optimization of carbon sequestration operations. We focus on results for Model 3.2, though optimizations were also performed using Model 3.1, and those findings will be discussed briefly at the end of this section.

3.3.1 Optimization setup

The goal of our optimization is to minimize a particular measure of the risk for CO\textsubscript{2} leakage. Various quantities could be considered, and some of these are discussed by Cameron and Durlofsky [12]. Here we use a relatively simple measure that corresponds to the quantity \( Z_{CO_2} \) defined in Eq. 3.16. Specifically, the objective function we seek to minimize, designated \( J \), is given by

\[
J(u) = \sum_{i \in D_{tl}} z_{CO_2,i}(T),
\]

(3.19)

where \( u \) designates the set of control variables, which in this case are the time-varying CO\textsubscript{2} injection rates for all wells, and \( T \) is the time corresponding to the end of the injection period (20 years). The target layer is layer 3 in this case, and \( D_{tl} \) denotes all grid blocks in that layer. The optimized controls will be designated \( u_{TPWL}^* \) if the optimization uses POD-TPWL for function evaluations, or \( u_{HFS}^* \) if the optimization uses high-fidelity simulation. By minimizing \( J \), the injected CO\textsubscript{2} is restricted to the lower part of the aquifer, away from the cap-rock. Thus, if there is a leak in the cap-rock, the injected CO\textsubscript{2} will not reach it quickly, which will provide time for remediation or the redesign of the storage project.

The well locations and general problem setup are as described in Section 3.2.2. The total (field) CO\textsubscript{2} injected at all times is constant and equal to about 8975 m\textsuperscript{3}/day at well conditions, which corresponds to about 1.5 million tonnes of CO\textsubscript{2} per year. The well injection rates are varied every five years, which means we have a total of 16 control variables. However, since we apply the constraint of fixed field-wide injection rate, the number of free variables is reduced to 12. As noted in Section 3.2.2, the full-order model includes 30,420 variables, and the POD-TPWL model is constructed
using \( l_p = l_z = 80 \). Derivatives from all five training runs are used in the POD-TPWL-based optimization.

The optimization algorithm used in this work is mesh adaptive direct search (MADS). MADS was developed by Audet and Dennis Jr [3], and that paper should be consulted for details. MADS is a derivative-free optimization procedure that involves stencil-based pattern search, in which the search directions do not coincide with the coordinate directions. At each iteration of MADS, up to \( 2n \) function evaluations (simulations) are performed, where \( n \) is the number of optimization variables (here \( n = 12 \)). The simulations can all be run in parallel. The point around which the search is conducted is the ‘best’ point found thus far in the optimization. Thus the search stencil moves through the parameter space from iteration to iteration. At an iteration where no improvement in the objective function value is achieved, the stencil is modified and/or reduced in size. Here we use the NOMAD package, which is an open source MADS implementation provided by Le Digabel [74]. In our optimizations, MADS is run in all cases for a total 200 function evaluations. Other termination criteria, such as a minimum MADS stencil size or minimum change in objective function value, could also be applied, but since our goal here is to illustrate a basic capability, we have not explored these other specifications.

### 3.3.2 Optimization results

The progress of the minimization for this case is shown in Fig. 3.17(a). The blue curve corresponds to a MADS optimization using POD-TPWL (which provides \( u_{\text{TPWL}}^* \)), and the dashed red curve to optimization using full-order AD-GPRS simulations (which provides \( u_{\text{HFS}}^* \)). The curves clearly differ, as they follow different paths to the local minimum, but the final objective function values are very close. The blue point (at 200 simulations) corresponds to the full-order result evaluated using the optimized solution \( u_{\text{TPWL}}^* \). This point is quite close to the blue curve, indicating that the POD-TPWL model provides an accurate estimate of the final objective function value \( J \) (defined in Eq. 3.19). The value of \( J \) corresponding to the initial guess is 3.33. Using optimization with the full-order model, \( J \) is reduced to 0.98, and using the POD-TPWL-based optimization it is reduced to 1.00 (evaluated using HFS). This corresponds to a decrease of 70%.
3.3. OPTIMIZATION OF CO$_2$ STORAGE OPERATIONS

Because MADS is a local optimization algorithm, it is useful to run it multiple times starting from different initial guesses to avoid having a solution corresponding to a poor local optimum. Optimization using POD-TPWL is appropriate for such an assessment since runtimes are very fast (over 100 times faster than with the full-order model). Results for five additional POD-TPWL-based optimizations, and the corresponding full-order validations using $u^*_{\text{TPWL}}$, are shown in Fig. 3.17(b). The dashed red curve is the same as in Fig. 3.17(a). One of these new solutions corresponds to a final validated (via HFS) $J$ value of 0.96, which is less than that achieved by optimization using the full-order model. By performing more POD-TPWL-based optimizations (starting from different initial guesses), it may be possible to reduce $J$ even further, though this was not attempted.

The initial-guess controls (designated $u_0$) along with the optimized controls from the best POD-TPWL-based optimization run are shown in Fig. 3.18. We see that the injection from Wells 2 and 3 is reduced, while that from Well 1 is increased. The overall injection volume from Well 4 appears somewhat comparable in the initial-guess and optimized scenarios, though the injection schedules clearly differ.

Initial guess and optimized results for the CO$_2$ distribution in layer 3, after 20 years of injection, are shown in Fig. 3.19. The white crosses denote the projected locations of the injection wells (which are in layer 13). The results in Fig. 3.19(b) correspond to the high-fidelity validation solution using $u^*_{\text{TPWL}}$. Consistent with the optimized
CHAPTER 3. POD-TPWL FOR CO₂ STORAGE OPERATIONS

Figure 3.18: Initial guess and optimized well controls (Model 3.2)

Figure 3.19: CO₂ molar fraction in layer 3 at the end of injection (20 years, Model 3.2)
injection rate schedules in Fig. 3.18, it is evident that considerably less CO$_2$ is injected in Well 3 in the optimized solution than in the initial-guess solution, and this leads to a reduction of the objective function $J$.

Cross-sectional views of the CO$_2$ distributions (initial guess and optimized) at each of the four wells are shown in Fig. 3.20. The white crosses indicate the well locations, and the target layer (layer 3) is designated by the white dashed lines. It is apparent from the plots that the cumulative CO$_2$ injected from Well 3 is reduced, while that from Well 1 is increased. Due to the connectivity of the permeability field near Well 1, the additional CO$_2$ tends to spread laterally rather than just move vertically, which limits the amount of CO$_2$ that enters layer 3.

The accuracy of the POD-TPWL solution in capturing the (optimal) CO$_2$ distribution is assessed in Fig. 3.21. Figures 3.21(a) and (c) present cross-sectional difference maps for CO$_2$ molar faction, for Wells 1 and 3, between the initial guess (with controls $u_0$) and optimized (with controls $u_{TPWL}^*$) solutions. Both of these results are generated via high-fidelity simulation, and the solutions are thus denoted by $HFS(u_0)$ and $HFS(u_{TPWL}^*)$. From these figures, the clear differences between the initial guess and optimized solutions are evident. Figures 3.21(b) and (d) display difference maps between the optimized POD-TPWL solution, denoted $TPWL(u_{TPWL}^*)$, and the full-order validated solution using controls $u_{TPWL}^*$. These differences are relatively small, and are clearly much less than the differences between the initial guess and optimized solutions. This again demonstrates that the POD-TPWL model can indeed provide sufficiently accurate function evaluations for the optimization problem considered here.

We also performed the same type of optimizations using Model 3.1. These results are quite analogous to those presented above for Model 3.2. Using Model 3.1, the objective function $J$ (with layer 6 as the target layer) was reduced from 5.03 to 2.40. The CO$_2$ distributions corresponding to the initial guess and optimized controls were again quite different, and the POD-TPWL solutions displayed accuracy comparable to that in the Model 3.2 results.
Figure 3.20: Cross-sectional views of CO$_2$ distribution at the end of injection (20 years, Model 3.2)
3.3. OPTIMIZATION OF CO$_2$ STORAGE OPERATIONS

Figure 3.21: Differences in initial guess and optimal solutions (left), and POD-TPWL model error (right), for Wells 1 and 3 (Model 3.2)
3.4 Summary

In this chapter, we extended the POD-TPWL reduced-order modeling procedure to enable the detailed simulation of the injection stage of a carbon storage operation. The use of multiple derivatives (i.e., multiple solutions around which linearization is performed) for POD-TPWL was introduced and evaluated. The POD-TPWL procedure is based on, and uses derivative information from, the molar variable compositional formulation in Stanford’s AD-GPRS subsurface flow simulator. The new POD-TPWL formulation was tested on two examples — a 3D aquifer characterized by a Gaussian log-permeability field, and a 3D channelized aquifer system. For both cases, the use of multiple derivatives was shown to improve POD-TPWL model accuracy for key quantities of interest. Runtime speedup of $O(100)$ was achieved for these cases. Our new POD-TPWL formulation was also applied to an optimization problem, where the objective was to minimize the total CO$_2$ at a target layer at the end of the injection period. A mesh adaptive direct search (MADS) algorithm was applied for this optimization. Detailed results were presented for the channelized system, and a reduction of about 70% in the total CO$_2$ in the target layer (relative to the initial-guess result) was achieved.
Chapter 4

Deep-Learning-Based ROM

In this chapter, we apply the embed-to-control (E2C) framework, a deep-learning-based formulation, to extend the applicability of oil-water reduced-order models. This treatment allows us to consider training data from a large number of runs, which enables us to handle large variations in control variables. We first review the governing equations for subsurface oil-water flow. The E2C formulation is then presented, and the correspondences between E2C and POD-TPWL are highlighted. Next, we present results for a 2D oil-water problem. We also provide a detailed error assessment for several key quantities. The detailed architectures for the encoder and decoder networks used in the E2C model are given in Appendix A. Additional simulation results with the E2C ROM are presented in Appendix B.

4.1 Governing Equations and POD-TPWL ROM

In this section, we present the equations for oil-water flow. Since the equations are very similar to those for the flow portion of the system considered in Chapter 2, the description here will be very succinct. We then provide a brief overview of the POD-TPWL ROM for this problem (including some new notation), which will allow us to draw analogies with the E2C ROM.
4.1.1 Governing equations

The governing equations for immiscible oil-water flow are close to those presented in Section 2.1.1, though here we neglect gravity. The flow equations in this case are:

$$\frac{\partial (\phi S_j \rho_j)}{\partial t} - \nabla \cdot (\lambda_j \rho_j k \nabla p) + \sum_w \rho_j q_j^w = 0,$$

(4.1)

where subscript $j$, again, denotes fluid phase ($j = o, w$ for oil and water). The rest of the variables are identical to those defined after Eq. 2.1. Again, capillary pressure is neglected. The oil-water model is completed by enforcing the saturation constraint $S_o + S_w = 1$. Because the system considered in this work is horizontal (in the $x - y$ plane), gravity effects do not appear.

Similar to our discussions in earlier chapters, after the equations are discretized with a standard finite volume formulation, the flow system is fully defined through the primary variables $p$ and $S_w$ in each grid block. The total number of variables in the discretized system is $2n_b$, where $n_b$ denotes the number of grid blocks in the model. We define $x_t = [p_t^T, S_t^T]^T \in \mathbb{R}^{2n_b}$ to be the state vector for the flow variables at a specific time step $t$, where $p_t \in \mathbb{R}^{n_b}$ and $S_t \in \mathbb{R}^{n_b}$ denote the pressure and saturation in every grid block at time step $t$.

Similar to Eqs. 2.8 and 3.2, the set of nonlinear algebraic equations representing the discretized fully-implicit system can be expressed as:

$$g(x_{t+1}, x_t, u_{t+1}) = 0,$$

(4.2)

where $g \in \mathbb{R}^{2n_b}$ is the residual vector (set of nonlinear algebraic equations) we seek to drive to zero. The time level indicators are now $t$ for the current time level and $t+1$ for the next time level, and they are now subscripted (instead of superscripted) for consistency with the embed-to-control equations shown later. The notation $u_{t+1} \in \mathbb{R}^{n_w}$ designates the well control variables, which can be any combination of bottom-hole pressures (BHPs) or well rates. Here $n_w$ denotes the number of wells in the system. In the work in this chapter, we operate production wells under BHP specifications and injection wells under rate specifications. Our treatments are general in this regard, and other control settings could also be applied. As noted in Chapters 2 and 3, and
as will be discussed later, both POD-TPWL and the deep-learning-based E2C ROM accelerate the solution of Eq. 4.2 by avoiding the test-time construction and iterative solution (with Newton’s method) of the high-dimensional \((2n_b \times 2n_b)\) system.

### 4.1.2 POD-TPWL formulation

Many deep-learning-based models involve treatments that are not directly analogous to those used in existing ROMs. Rather, they entail machine-learning methods that derive from image classification, language recognition, or other non-PDE-based applications. Our E2C ROM is somewhat different in this sense, as its three main components are directly analogous to those used in POD-TPWL. Because POD-TPWL has been extensively developed for subsurface flow applications, we believe it is worthwhile to discuss the correspondences between the POD-TPWL and E2C ROMs. Since detailed descriptions of POD-TPWL have already been given in Chapters 2 and 3, here we only provide a high-level overview of POD-TPWL that is geared to enable comparisons with E2C.

As discussed earlier, POD-TPWL and other POD-based ROMs involve an offline (here referred to as train-time) stage and an online (test-time) stage. During the offline stage, a number of training simulation runs are performed using a full-order simulator (AD-GPRS in this work). The goal here is to predict test-time results with varying (new) well control sequences. Therefore, during training runs, we apply different well control sequences \(U = [u_1, \ldots, u_{N_{ctrl}}] \in \mathbb{R}^{n_w \times N_{ctrl}}, \) where \(u_k \in \mathbb{R}^{n_w}, k = 1, \ldots, N_{ctrl},\) contains the settings for all wells at control step \(k,\) and \(N_{ctrl}\) denotes the total number of control steps in a training run. There are many fewer control steps than time steps in a typical simulation (in our examples we have 20 control steps and around 100 time steps). As discussed in Chapters 2 and 3, state variables in all grid blocks (solution snapshots) and derivative matrices are saved at each time step in the training runs. At test-time, simulations with control sequences that are different from those of the training runs are performed. Information saved from the training runs is used to (very efficiently) approximate test solutions.

Conceptually, POD-TPWL entails (1) projection from a high-dimensional space to a low-dimensional subspace, (2) linear approximation of the dynamics in the low-dimensional subspace, and (3) projection back to the high-dimensional space. Similar
to Eq. 2.12 for the coupled flow-geomechanics system, and Eq. 3.6 for the CO$_2$-water system, a projection matrix $\Phi \in \mathbb{R}^{2n_b \times l_\xi}$ is constructed here from the singular value decomposition (SVD) of the solution snapshot matrices for the oil-water system. As explained earlier, given $\Phi$, the high-dimensional states $x \in \mathbb{R}^{2n_b}$ can be represented in terms of the low-dimensional variable $\xi \in \mathbb{R}^{l_\xi}$ using

$$x \approx \Phi \xi,$$  \hspace{1cm} (4.3)

where $l_\xi$ is the dimension of the reduced space, with $l_\xi \ll n_b$. The SVD and subsequent projections are performed separately for the pressure and saturation variables. Because $\Phi$ is orthonormal, we also have $\xi = \Phi^T x$.

Before discussing the POD-TPWL approximation in low-dimensional space, we first show the linearization in high dimension. Our notation is slightly different here than in previous chapters. This notation is introduced to enable direct comparisons with E2C. The TPWL formulation (with the POD representation for states, $x = \Phi \xi$, applied to the right-hand side) can be expressed as

$$J_{i+1} \dot{x}_{t+1} = J_{i+1} \Phi \xi_{i+1} - [A_{i+1} \Phi (\xi_t - \xi_i) + B_{i+1} (u_{i+1} - u_{i+1})],$$  \hspace{1cm} (4.4)

where

$$J_{i+1} = \frac{\partial g^{i+1}}{\partial x^{i+1}} \in \mathbb{R}^{2n_b \times 2n_b}, \quad A_{i+1} = \frac{\partial g^{i+1}}{\partial x^i} \in \mathbb{R}^{2n_b \times 2n_b}, \quad B_{i+1} = \frac{\partial g^{i+1}}{\partial u^{i+1}} \in \mathbb{R}^{2n_b \times n_w}.$$  \hspace{1cm} (4.5)

Here the subscripts $t$ and $t + 1$ denote time steps in the test run, while the subscripts $i$ and $i + 1$ designate time steps in the training simulations. Note that Eq. 4.4 differs slightly from Eqs. 2.10 and 3.4 since the time step designations are now subscripted. The variable $\xi_t$ is the projection of the true (high-order) solution of Eq. 4.2 at time step $t$. The variable $\dot{x}_{t+1} \in \mathbb{R}^{2n_b}$ is distinct from $x_{t+1}$, in that it represents the full-order variable at time step $t + 1$ approximated through linearization instead of via solution of the full-order system (Eq. 4.2). From here on in this chapter, we will use variables without ‘hats’ to denote the true high-order solution (e.g., $x$) or the true solution projected with matrix $\Phi$ (e.g., $\xi = \Phi^T x$). And, we will use variables with ‘hats’ ($\dot{x}$ and $\dot{\xi}$) to designate solutions approximated (either reconstructed or predicted, as will
be explained in detail later) by the ROM. The variables $u_{t+1}, u_{i+1} \in \mathbb{R}^{n_w}$ are the well settings at time step $t + 1$ and $i + 1$ — these are prescribed by the user or specified by an optimization algorithm, as in the developments in Chapters 2 and 3.

Applying the POD representation on the left-hand side and constraint reduction (projection) on both sides of Eq. 4.4, the solution approximation in low-dimensional space, after some rearrangement, is given by

$$\hat{\xi}_{t+1} = \xi_{i+1} - (J_{r_{i+1}}^r)^{-1}[A_{r_{i+1}}^r(\xi_t - \xi_i) + B_{r_{i+1}}^r(u_{t+1} - u_{i+1})],$$

(4.6)

with the reduced derivative matrices defined as

$$J_{r_{i+1}}^r = (\Psi_{i+1})^T J_{i+1} \Phi, \quad A_{r_{i+1}}^r = (\Psi_{i+1})^T A_{i+1} \Phi, \quad B_{r_{i+1}}^r = (\Psi_{i+1})^T B_{i+1}.$$  

(4.7)

Here $J_{r_{i+1}}^r \in \mathbb{R}^{\ell \times \ell}$, $A_{r_{i+1}}^r \in \mathbb{R}^{\ell \times \ell}$ and $B_{r_{i+1}}^r \in \mathbb{R}^{\ell \times n_w}$. The matrix $\Psi_{i+1}$ denotes the constraint reduction matrix at time step $i + 1$. The variable $\hat{\xi}_{t+1} \in \mathbb{R}^{\ell}$ represents the reduced variable approximated through linearization at time step $t + 1$. Note that Eq. 4.6 here closely resembles Eqs. 2.19 and 3.10. However, in those equations the equivalent of $\hat{\xi}_t$ appears on the right-hand side, while $\xi_i$ appears in Eq. 4.6.

During the online stage (test-time), we do not know $\xi_t$ (the projected true solution of Eq. 4.2 at time step $t$). Rather, we have $\hat{\xi}_t$, the reduced variable approximated through linearization at time step $t$ (computed from Eq. 4.6 at the previous time step). Therefore, at test-time, Eq. 4.6 becomes

$$\hat{\xi}_{t+1} = \xi_{i+1} - (J_{r_{i+1}}^r)^{-1}[A_{r_{i+1}}^r(\hat{\xi}_t - \xi_i) + B_{r_{i+1}}^r(u_{t+1} - u_{i+1})].$$

(4.8)

Note that $\hat{\xi}_t$ now appears on the right-hand side instead of $\xi_t$. This equation now corresponds directly with Eqs. 2.19 and 3.10. At test-time, the training ‘point,’ around which linearization is performed (this point defines $i$ and $i + 1$), is determined using a point-selection procedure. This point selection depends on $\hat{\xi}_t$ (as described in Section 3.1.4), so the reduced derivative matrices $J_{r_{i+1}}^r$, $A_{r_{i+1}}^r$ and $B_{r_{i+1}}^r$ can all be considered to be functions of $\hat{\xi}_t$. In the last step of POD-TPWL, the approximated solutions are projected back to the full-order space through application of $\hat{x} = \Phi \hat{\xi}$, where $\hat{\xi} \in \mathbb{R}^{\ell}$ is the approximated test state in the subspace, and the variable $\hat{x} \in$
\( \mathbb{R}^{2n_b} \) is the corresponding approximated state in the full-dimensional space.

Each of the above-mentioned steps in POD-TPWL can be viewed in terms of an optimization (in some cases heuristic), as we now consider. The projection matrix \( \Phi \) is constructed using the POD procedure. This has the well-known property that the resulting basis matrix minimizes a projection error \( e_{\text{proj}} \), defined as

\[
e_{\text{proj}} = \| x - \Phi \Phi^T x \|_2^2,
\]

(4.9)

where \( x \in \mathbb{R}^{2n_b} \) is the full-order state variable.

In addition, as discussed by He and Durlofsky [47], the constraint reduction error can be defined as

\[
e_{\text{cr}} = \| \xi - \Phi \hat{\xi} \|_{\Theta}^2,
\]

(4.10)

where \( \xi \) corresponds to the solution \( \hat{x}_{t+1} \) in Eq. 4.4 (before constraint reduction is applied); this variable was denoted as \( x_2 \) in [47]. The variable \( \hat{\xi} \) corresponds to the solution \( \hat{\xi}_{t+1} \) in Eq. 4.6 (after constraint reduction is applied) and was expressed as \( \xi_3 \) in [47]. The notation \( \| \cdot \|_{\Theta} \) is a norm defined as \( \| e \|_{\Theta} = \sqrt{e^T \Theta e} \), with \( e \in \mathbb{R}^{2n_b} \) and \( \Theta \in \mathbb{R}^{2n_b \times 2n_b} \), where \( \Theta \) is a symmetric positive definite matrix. The optimal constraint reduction matrix \( \Psi \) can be determined by minimizing the constraint reduction error, i.e.,

\[
\Psi = \arg \min_{\Psi} e_{\text{cr}}.
\]

(4.11)

If the matrix \( \Theta \) is defined as \( J^T J \) then, following Eqs. 21 through 27 in [47], we arrive at the least-squares Petrov-Galerkin projection, i.e.,

\[
\Psi = J \Phi.
\]

(4.12)

This treatment, which as we see is optimal in a particular norm, is now routinely applied in POD-TPWL (and was used in Chapters 2 and 3).

Thus it is evident that, in the POD-TPWL procedure, the low-dimensional state representation and the constraint reduction procedure are based on two distinct optimizations, with (separate) objective functions \( e_{\text{proj}} \) and \( e_{\text{cr}} \). The remaining aspect of POD-TPWL to be considered is point selection. Different point-selection strategies have been used for different applications, and these typically include a heuristic
4.2. EMBED-TO-CONTROL FORMULATION

component (as is the case for the multiple-derivative point selection introduced in Section 3.1.4). These procedures do, however, entail the minimization of a ‘distance’ metric, which quantifies the distance (in an application-specific sense) between the current test point and a large set of training-run points. Thus, this step also entails an optimization. As we will see, these POD-TPWL component optimizations correspond to the loss function minimization that will be applied in the embed-to-control framework. A key difference, however, is that in the E2C framework all of the steps are optimized together, rather than separately as in POD-TPWL.

4.2 Embed-to-Control Formulation

In this section, we develop an embed-to-control ROM that includes physical constraints. Analogies to POD-TPWL are established for the various E2C components. The E2C model presented here generally follows that developed by Watter et al. [113], though several important modifications are introduced, as will be discussed below.

4.2.1 E2C overview

The embed-to-control framework entails three processing steps: an encoder or inference model that projects the system variables from a high-dimensional space to a low-dimensional subspace (referred to here as the latent space), a linear transition model that approximates system dynamics in low-dimension, and a decoder or generative model that projects solutions back to high-dimensional (full-order) space. The E2C framework originally proposed by Watter et al. [113] used a VAE architecture for both the encoder and decoder procedures, which allowed them to account for uncertainty in predictions. In the formulation here, the VAE architecture is reduced to an auto-encoder (AE) architecture, since we are considering deterministic systems.

We note that the auto-encoder (AE) architecture is commonly used for semantic segmentation [95], where each pixel of the image is associated with a class label, and for depth prediction [28], where the 3D geometry of a scene is inferred from a 2D image. In the context of subsurface flow simulation, AE architectures have been used to construct surrogate simulation models as an image-to-image regression, where the input images are reservoir properties (e.g., permeability fields) and the outputs are
state variables [127, 83].

Figure 4.1 displays the overall workflow for our embed-to-control model. The pressure field $\mathbf{p}_i \in \mathbb{R}^{n_b}$ is the only state variable shown in this illustration (the subscript $i$, distinct from $t$, denotes the time steps in a training run), though our actual problem also includes the saturation field $\mathbf{S}_i \in \mathbb{R}^{n_b}$. Additional state variables would appear in more general settings (e.g., displacements if a coupled flow-geomechanics model is considered).

![Figure 4.1: Embed-to-control (E2C) overview](image)

Box 1 in Fig. 4.1 displays pressure snapshots $\mathbf{p}_i \in \mathbb{R}^{n_b}, i = 1, \ldots, N_s$ in the full-order space, where $N_s$ is the total number of snapshots. The notation $Q^{\text{enc}}_\phi$ in Funnel 2 denotes the encoder, which projects the full space into a latent space, with $\phi$ representing all of the ‘learnable’ parameters in the encoder. The variable $\mathbf{z}^p_i \in \mathbb{R}^{l_z}$ in Box 3 is the latent variable for pressure, with $l_z$ the dimension of the latent space.

In Box 3, the test simulation results are approximated in the latent space with a linear transition model. The variable $\mathbf{z}^p_0 \in \mathbb{R}^{l_z}$ denotes the initial latent state for a test run, and $\mathbf{u}_t \in \mathbb{R}^{n_w}, t = 1, \ldots, N_{\text{ctrl}}$ designates the control sequence for a test run, with $n_w$ the number of wells (as noted previously), the subscript $t$ indicates time step in the test run, and $N_{\text{ctrl}}$ is the number of control steps in the test run. The
linear transition model $\hat{Q}_t^{\text{trans}}$ ($\psi$ denotes the learnable parameters) takes $z^p_0 \in \mathbb{R}^{l_z}$ and $u_t \in \mathbb{R}^{n_u}$ as input, and outputs $z^p_t \in \mathbb{R}^{l_z}$, $t = 1, \ldots, N_{te}$ sequentially, where $N_{te}$ is the total number of time steps in a test run. The decoder $P^{\text{dec}}_t$ (indicated by Funnel 4, with $\theta$ representing all of the learnable parameters in the decoder) then projects the variable $z^p_t$ back to the full-order state $p_t \in \mathbb{R}^{n_b}$, as shown in Box 5.

The embed-to-control ROM incorporates the control variable $u_t \in \mathbb{R}^{n_u}$ naturally in the framework. This is an important distinction relative to the AE-LSTM-based ROM developed by Gonzalez and Balajewicz [43], where system controls were not included in the model. In the following subsections, the three main components of the embed-to-control framework, the encoder, the linear transition model, and the decoder, will be discussed in detail. A loss function with physical constraints, along with E2C implementation details, will also be presented.

### 4.2.2 Encoder component

The encoder provides a low-dimensional representation of the full-order state variables. In contrast to the original embed-to-control implementation by Watter et al. [113], here we adopt an AE instead of a VAE architecture. With this treatment only the mean values of the latent variables are estimated, not the variances. Also, we do not require a sampling process in the latent space. Therefore, at train-time, the encoder can be simply expressed as

$$z_t = Q^{\text{enc}}_t(x_t),$$

where $Q^{\text{enc}}_t$ represents the encoder, as explained previously. The variable $x_t \in \mathbb{R}^{2n_b}$ is the full-order state variable at time step $t$, and $z_t \in \mathbb{R}^{l_z}$ is the corresponding latent variable, with $l_z$ the dimension of the latent space. In the examples presented later, we consider a 2D $60 \times 60$ oil-water model (which means the full-order system is of dimension 7200), and we set $l_z = 50$. Note that Eq. 4.13 is analogous to Eq. 4.3 in the POD-TPWL procedure, except the linear projection in POD is replaced by a nonlinear projection $Q^{\text{enc}}_t$ in the encoder. Following the convention described earlier, we use variables without a ‘hat’ to denote (projected) true solutions of Eq. 4.2, which are available from training runs. Variables with a hat designate approximate solutions...
provided by the test-time ROM.

The detailed layout of the encoder in the E2C model is presented in Fig. 4.2. During training, sequences of pressure and saturation snapshots are fed through the encoder network, and sequences of latent state variables $z_t \in \mathbb{R}^{l_z}$ are generated. The encoder network used here is comprised of a stack of four encoding blocks, a stack of three residual convolutional (resConv) blocks, and a dense layer. The encoder in Fig. 4.2 is more complicated (i.e., it contains resConv blocks and has more convolutional layers) compared to those used in [113]. A more complicated structure may be needed here because, compared to the prototype planning tasks addressed in [113] (e.g., cart-pole balancing, and three-link robotic arm planning), proper representation of PDE-based pressure and saturation fields requires feature maps from a deeper network.

![Encoder Network Diagram](image)

Figure 4.2: Encoder layout

Similar to the CNN-PCA proposed by Liu et al. [79], which uses the filter operations in CNN to capture the spatial correlations that characterize geological features, the embed-to-control framework uses stacks of convolutional filters to represent the spatial distribution of the pressure and saturation fields determined by the underlying governing equations. Earlier implementations with AE/VAE-based ROMs [77, 43]
have demonstrated the potential of convolutional filters to capture such fields in fluid dynamics problems. Thus, our encoder network is mostly comprised of these convolutional filters (in the form of 2D convolutional layers, i.e., conv2D layer [76]). More detail on the encoder network is provided in Table A.1 of Appendix A.

The first step in an encoding block is a convolution operation, which can also be viewed as a linear filter. Following the expression in [79], the mathematical formulation of linear filtering is

\[
F_{i,j}(x) = \sum_{p=-n}^{n} \sum_{q=-n}^{n} w_{p,q} x_{i+p,j+q} + b,
\]

(4.14)

where \( x \) is the input state map, subscripts \( i \) and \( j \) denote \( x \) and \( y \) coordinate direction indices, \( w \) represents the weights of a linear filter (template) of size \((2n+1) \times (2n+1)\), \( F_{i,j}(x) \) designates the filter response map (i.e., feature map) for \( x \) at spatial location \((i, j)\), and \( b \) is a scalar parameter referred to as bias. Note that there are typically many filters associated with a conv2D layer, and the filter response map, which collects all of these operations, is thus a third-order tensor. The output filter response maps are then passed through a batch normalization (batchNorm) layer [53], which applies normalization operations (shifts the mean to zero and rescales by the standard deviation) for each subset of training data. A batchNorm operation is a crucial step in the efficient training of deep neural networks, since it renders the learning process less sensitive to parameter initialization, which means a larger initial learning rate can be used. The nonlinear activation function ReLU (rectified linear unit, \( \max(0,x) \)) [41] is applied on the normalized filter response maps to give a final response (output) of the encoder block. This nonlinear response is referred to as the ‘activation’ of the encoding block. The conv2D-batchNorm-ReLU architecture (with variation in ordering) is a standard processing step in CNNs. An illustration of the encoding block structure can be found in Fig. A.1(a) of Appendix A.

To properly incorporate feature maps capable of representing the spatial pressure and saturation distributions, as determined by the underlying governing equations, a deep neural network with many stacks of convolutional layers is required. Deep neural networks are, however, difficult to train, mostly due to the gradient vanishing issue [40]. By this we mean that gradients of the loss function with respect to the
model parameters (weights of the filters) become vanishingly small, which negatively impacts training. He et al. [50] addressed this issue by creating an additional identity mapping, referred to as resNet, that bypasses the nonlinear layer. Following the idea of resNet, we add a stack of resConv blocks to the encoder network to deepen the network while mitigating the vanishing-gradient issue. The nonlinear layer in the resConv block still generally follows the conv2D-batchNorm-ReLu architecture. See Fig. A.1(c) in Appendix A for a visual representation of the resConv block.

Similar to that of the encoding block, the output of resConv blocks is a stack of low-dimension feature maps. This stack of feature maps is ‘flattened’ to a vector (which is still a relatively high-dimensional vector due to the large number of feature maps), and then input to a dense layer. A dense (fully-connected) layer is simply a linear projection that maps a high-dimensional vector to a low-dimensional vector.

The overall architecture of the encoder network used here differs from that constructed by Zhu and Zabaras [127] in three key aspects. First, resNet is used in our encoder while they used denseNet [52] to mitigate the vanishing-gradient issue. Another key distinction is that the encoder (and the decoder) in [127] do not include the dense layer at the end, which means the encoder outputs a stack of feature maps at the end. A large number of feature maps (i.e., a tall but relatively thin third-order tensor) would be too high-dimensional for the sequential linear operations subsequently performed by the linear transition model. Finally, Zhu and Zabaras [127] adopted a U-Net [95] architecture, which is reasonable when the output of the encoder-decoder (e.g., pressure field) differs from the input (e.g., permeability map), as was the case in their setting. However, the U-Net architecture is inappropriate and may lead to over-fitting when the inputs and outputs are of the same type (i.e., both are pressure and saturation fields), as is the case here.

The encoder (and decoder) in the embed-to-control ROM is analogous to the POD representation used in POD-TPWL. As noted earlier, the basis matrix $\Phi$ constructed via SVD of the snapshot matrices has the feature that it minimizes $e_{proj}$ in Eq. 4.9. In the context of the encoder, a reconstruction loss $\mathcal{L}_R$, which is similar to $e_{proj}$ for POD, is computed. Conceptually, the ‘best’ $Q_{\Phi}^{enc}$ is found by minimizing $\mathcal{L}_R$. However, as mentioned earlier, the optimization applied for the embed-to-control model involves all three processing steps considered together, so $\mathcal{L}_R$ is not minimized separately.
4.2.3 Linear transition model

The linear transition model evolves the latent variable from one time step to the next, given the controls. Fig. 4.3 shows how the linear transition model is constructed and evaluated during the offline stage (train-time). The inputs to the linear transition model include the latent variable for the current state $z_t \in \mathbb{R}^{l_z}$, the current step control $u_{t+1} \in \mathbb{R}^{n_w}$, and time step size $\Delta t$. The model outputs the predicted latent state for the next time step $\hat{z}_{t+1} \in \mathbb{R}^{l_z}$. We reiterate that $\hat{z}_{t+1}$ represents the output of the linear transition model. The structure of the linear transition model, which generally follows that in [113], is comprised of a stack of three transformation (trans) blocks and two dense layers. The trans block follows a dense-batchNorm-ReLU architecture (dense represents a dense layer), which is considered a standard processing step for fully-connected networks. The trans block architecture is shown in Fig. A.1(c) in Appendix A. The variables $z_t$ and $\Delta t$ are first fed into the trans blocks. The final activation vector of the trans blocks, $h_{\psi,\phi}^{\text{trans}}$, is then used to construct the linearization matrices $A_t \in \mathbb{R}^{l_z \times l_z}$ and $B_t \in \mathbb{R}^{l_z \times n_w}$ through two separate dense layers. Matrices $A_t$ and $B_t$ are then combined with the latent variable for the current state $z_t$ and current step control $u_{t+1}$ to predict the latent variable at the next time step $\hat{z}_{t+1}$.

![Figure 4.3: Linear transition model layout](image)

The optimization applied to determine the parameters for the linear transition model is again analogous to a key step in POD-TPWL. In POD-TPWL, the goal is essentially to minimize the difference between the predicted reduced state $\hat{\xi}^{t+1}$ and the projected true state $\xi^{t+1}$. This is achieved, in part, by determining the optimal
CHAPTER 4. DEEP-LEARNING-BASED ROM

constraint reduction matrix $\Psi$, as described in Eqs. 4.10 and 4.11. Given this optimal $\Psi$ matrix, the matrices appearing in the POD-TPWL evolution equation (Eq. 4.6), namely $J_{r+1}^r$, $A_{r+1}^r$ and $B_{r+1}^r$, are all fully defined. As discussed earlier, point-selection represents another (heuristic) optimization that appears in POD-TPWL. Similarly, in the embed-to-control formulation, a transition loss $L_T$ is computed by comparing $\hat{z}_{t+1}$ with $z_{t+1}$, where $\hat{z}_{t+1}$ is the output from the linear transition model, and $z_{t+1}$ is the state projected by the encoder at time step $t + 1$. The transition loss contributes to the total loss function, which is minimized during the offline stage.

The linear transition model at train-time can also be represented as

$$\hat{z}_{t+1} = \hat{Q}^{\text{trans}}_t(z_t, u_{t+1}, \Delta t),$$

(4.15)

where $\Delta t$ is the time step size, the function $\hat{Q}^{\text{trans}}$ is the linear transition model as previously defined ($\psi$ denotes all the learnable parameters within the model), and $\hat{z}_{t+1} \in \mathbb{R}^{l_z}$ denotes the latent variable at $t + 1$ predicted by the linear transition model. To be more specific, Eq. 4.15 can be expressed as

$$\hat{z}_{t+1} = A_t(z_t, \Delta t)z_t + B_t(z_t, \Delta t)u_{t+1},$$

(4.16)

where $A_t \in \mathbb{R}^{l_z \times l_z}$ and $B_t \in \mathbb{R}^{l_z \times n_w}$ are matrices. Consistent with the expressions in [113], these matrices are given by

$$\text{vec}[A_t] = W_A h^{\text{trans}}_{\psi'}(z_t, \Delta t) + b_A,$$

(4.17)

$$\text{vec}[B_t] = W_B h^{\text{trans}}_{\psi'}(z_t, \Delta t) + b_B,$$

(4.18)

where vec denotes vectorization, so vec$[A_t] \in \mathbb{R}^{(l_z^2) \times 1}$ and vec$[B_t] \in \mathbb{R}^{(l_z n_w) \times 1}$. The variable $h^{\text{trans}}_{\psi'} \in \mathbb{R}^{n_{\text{trans}}}$ represents the final activation output after three transformation blocks (which altogether are referred to as the transformation network). The $\psi'$ in Eqs. 4.17 and 4.18 is a subset of $\psi$ in Eq. 4.15, since the latter also includes parameters outside the transformation network. Here $W_A \in \mathbb{R}^{l_z^2 \times n_{\text{trans}}}$, $W_B \in \mathbb{R}^{(l_z n_w) \times n_{\text{trans}}}$, $b_A \in \mathbb{R}^{(l_z^2) \times 1}$, and $b_B \in \mathbb{R}^{(l_z n_w) \times 1}$, where $n_{\text{trans}}$ denotes the dimension of the transformation network. We set $n_{\text{trans}} = 200$ in the model tested here.

During the online stage (test-time) the linear transition model is slightly different,
4.2. EMBED-TO-CONTROL FORMULATION

since the latent variable fed into the model ($\hat{z}_t \in \mathbb{R}^{l_z}$) is predicted from the last time step. Therefore, at test-time, Eq. 4.16 becomes

$$\hat{z}_{t+1} = A_t(\hat{z}_t, \Delta t)\hat{z}_t + B_t(\hat{z}_t, \Delta t)u_{t+1}. \quad (4.19)$$

Note the only difference is that $z_t$ on the right-hand side of Eq. 4.16 is replaced by $\hat{z}_t$ in Eq. 4.19.

The test-time formulation of the linear transition model is directly analogous to the linear representation step in POD-TPWL. In POD-TPWL, since the training step $i$ (and thus $i+1$) is determined based on the point-selection calculation involving $\hat{\xi}_t$, the matrices appearing in the online expression (Eq. 4.8) can be considered to be functions of $\hat{\xi}_t$. After some reorganization, Eq. 4.8 can then be written as

$$\hat{\xi}^{i+1} = A_{t}^{\text{TPWL}}(\hat{\xi})\hat{\xi}_t + B_{t}^{\text{TPWL}}(\hat{\xi})u_{t+1} + c_{t}^{\text{TPWL}}, \quad (4.20)$$

where

$$A_{t}^{\text{TPWL}} = -(J_{r}^{i+1})^{-1}A_{r}^{i+1}, \quad B_{t}^{\text{TPWL}} = -(J_{r}^{i+1})^{-1}U_{r}^{i+1},$$

$$c_{t}^{\text{TPWL}} = -A_{t}^{\text{TPWL}}\xi - B_{t}^{\text{TPWL}}u^{i+1} + \xi^{i+1}. \quad (4.21)$$

Thus we see that Eq. 4.19 for the online stage of the embed-to-control formulation is of the same form as Eq. 4.20 for the online stage of POD-TPWL. The key difference is that matrices $A_t$ and $B_t$ in E2C are determined by a deep-learning model instead of being constructed from derivative matrices from training runs. Note also that $c_t$ does not appear in the E2C formulation, since this representation does not entail expansion around nearby solutions.

4.2.4 Decoder component

The decoder is similar to the encoder and can be represented as

$$\hat{x}_t = P_{\theta}^{\text{dec}}(z_t), \quad (4.22)$$

where $P_{\theta}^{\text{dec}}$ is the decoder as previously defined. The variable $\hat{x}_t \in \mathbb{R}^{2n_b}$ denotes the reconstructed state variable at time step $t$ (which is distinct from the high-fidelity state variable $x_t \in \mathbb{R}^{2n_b}$ from the training snapshots), though the input to the decoder
\( z_t \in \mathbb{R}^{l_z} \) is the latent variable determined from the encoding of \( x_t \). If the input is instead \( \hat{z}_{t+1} \in \mathbb{R}^{l_z} \), which is the latent variable predicted at time step \( t + 1 \) by the linear transition model, Eq. 4.22 becomes

\[
\hat{x}_{t+1} = P_{\phi}^{\text{dec}}(\hat{z}_{t+1}),
\]

where \( \hat{x}_{t+1} \) is the predicted state variable at time step \( t + 1 \). Note that Eq. 4.22 only appears in the train-time procedure (to compute reconstructed states), while Eq. 4.23 has the same form at both train-time and test-time.

The detailed structure of the decoder is shown in Fig. 4.4. Latent variables predicted by the linear transition model (at time step \( t + 1 \)) are fed to the decoder network as input, and the predicted high-dimensional states are output. The architecture of the decoder is analogous to that of the encoder except it is in reversed order (which is not surprising since the decoder is conducting the inverse operation). The decoder here is comprised of a dense layer, a stack of three resConv blocks, a stack of four decoding blocks, and a conv2D layer. The dense layer converts a low-dimensional latent vector to a stack of feature maps (after reshaping). The feature maps are expanded while going through stacks of resConv blocks and decoding blocks. The spatial distributions of the pressure and saturation fields are sequentially ‘extracted’ from the feature maps as we proceed downstream in the decoder. The conv2D layer at the end converts the expanded feature maps to pressure and saturation fields as the final outputs. More detail on the encoder is provided in Table A.2 in Appendix A. The layout of the decoding block is shown in Fig. A.1(b) of Appendix A.

To determine the learnable parameters \( \theta \) in the decoder, a prediction loss \( L_{PD} \) is minimized (along with the other losses) in the offline process. More details on this optimization will be presented later.

### 4.2.5 Loss function with physical constraints

We have described each of the components of the embed-to-control framework. We now explain how the model parameters are determined during the offline stage. The parameters for the embed-to-control framework are \( \phi, \psi \), and \( \theta \) for the encoder, linear transition model, and decoder, respectively. The objective function to be minimized
4.2. EMBED-TO-CONTROL FORMULATION

is the total loss function that quantifies the overall performance of the model in predicting the output state variables.

We have briefly introduced the reconstruction loss ($\mathcal{L}_R$), the linear transition loss ($\mathcal{L}_T$), and the prediction loss ($\mathcal{L}_{PD}$), which comprise major components of the total loss function. To be more specific, the reconstruction loss for a training data point $i$ can be expressed as

$$\mathcal{L}_R(i) = \{||x_t - \hat{x}_t||^2\}_{i},$$

where $i = 1, \ldots, N_t$, with $N_t$ denoting the total number of data points generated in the training runs. Note that $N_t = N_s - n_{\text{train}}$, where $N_s$ is the total number of snapshots in the training runs and $n_{\text{train}}$ is the number of training simulations performed. Here $N_t$ and $N_s$ differ because, for a training simulation containing $N_{\text{tr}}$ snapshots, only $N_{\text{tr}} - 1$ data points can be collected (since pairs of states, at sequential time steps, are required). The variable $x_t$ is the state variable at time step $t$ from a training simulation, and $\hat{x}_t = P_{\theta}^{\text{dec}}(z_t) = P_{\theta}^{\text{dec}}(Q_{\theta}^{\text{enc}}(x_t))$ denotes the states reconstructed by the encoder and decoder.
The linear transition loss for training point \( i \) is similarly defined as

\[
(L_T)_i = \{ \| z_{t+1} - \hat{z}_{t+1} \|^2 \}_i,
\]

where \( z_{t+1} = Q_{\phi}^{enc}(x_{t+1}) \) is the latent variable encoded from the full-order state variable at \( t + 1 \), and the variable \( \hat{z}_{t+1} = Q_{\psi}^{trans}(z_t, u_{t+1}, \Delta t) \) denotes the latent variable predicted by the linear transition model. Finally, the prediction loss for training point \( i \) is defined as

\[
(L_{PD})_i = \{ \| x_{t+1} - \hat{x}_{t+1} \|^2 \}_i,
\]

where \( x_{t+1} \) designates the state variable at time step \( t + 1 \) from the training simulations, and \( \hat{x}_{t+1} = P_{\theta}^{dec}(\hat{z}_{t+1}) \) represents the full-order state variable predicted by the ROM. The data mismatch loss is the sum of these losses averaged over all training data points,

\[
L_d = \frac{1}{N_t} \sum_{i=1}^{N_t} (L_R)_i + (L_{PD})_i + \lambda(L_T)_i,
\]

where \( \lambda \) is a weight term.

The ROM as described up to this point is a purely data driven model, i.e., the goal of the model is to minimize the pixel-wise difference between the E2C output and the high-fidelity solution (the HFS is taken as the ‘true’ reference solution). Physical behavior is, to some extent, inferred by E2C from the input pressure and saturation snapshots, but it is not explicitly enforced. If the ROM is trained using the loss function \( L_d \) given in Eq. 4.27, unphysical effects can, however, be observed. This is illustrated in Fig. 4.5, where we show predictions for the pressure field at a particular time (the problem setup will be described in detail in Section 4.3). The high-fidelity solution is shown in Fig. 4.5(a), and the E2C pressure field based solely on \( L_d \) appears in Fig. 4.5(c). Although the two results are visually similar, the difference map in Fig. 4.5(e) indicates that the E2C result is not sufficiently smooth, and relatively large errors appear at some spatial locations. This could have a significant impact on well rate predictions, which are an essential ROM output.

To address this issue, we combine the loss for data mismatch with a loss function based on flow physics. Specifically, we seek to minimize the inconsistency in flux between each pair of adjacent grid blocks. Extra weight is also placed on key well
4.2. EMBED-TO-CONTROL FORMULATION

(a) High-fidelity solution (HFS\textsubscript{test})

(b) High-fidelity solution (HFS\textsubscript{test})

(c) ROM solution without $L_p$

(d) ROM solution with $L_p$

(e) $|\text{HFS}_{\text{test}} - \text{ROM}_{\text{test}}|$ without $L_p$
   (max error 97 psi)

(f) $|\text{HFS}_{\text{test}} - \text{ROM}_{\text{test}}|$ with $L_p$
   (max error 16 psi)

Figure 4.5: Pressure field predictions with and without $L_p$ (all colorbars in units of psi)
quantities. We consider both reconstruction (at time step $t$) and prediction (at time step $t + 1$). Thus we define the physics-based loss for each data point, $(\mathcal{L}_p)_i$, as

$$
(\mathcal{L}_p)_i = \{k \cdot [\nabla \mathbf{p}_t \cdot \nabla \mathbf{p}_t]_{\text{recon}} + (\nabla \mathbf{p}_{t+1} \cdot \nabla \mathbf{p}_{t+1})_{\text{pred}}\}_i^2 + \gamma \{((\mathbf{q}^w_t - \hat{\mathbf{q}}^w_t)_{\text{recon}} + (\mathbf{q}^w_{t+1} - \hat{\mathbf{q}}^w_{t+1})_{\text{pred}}\}_i^2.
$$

(4.28)

Here $\mathbf{p}_t, \mathbf{p}_{t+1} \in \mathbb{R}^{n_b}$ are the pressure fields at time steps $t$ and $t + 1$ from the training data, which are components of the state variables $\mathbf{x}_t$ and $\mathbf{x}_{t+1}$, and $\hat{\mathbf{p}}_t, \hat{\mathbf{p}}_{t+1} \in \mathbb{R}^{n_b}$ represent the ROM pressure reconstruction (at time step $t$, defined after Eq. 4.24) and prediction (at time step $t + 1$, defined after Eq. 4.26). The variables $\mathbf{q}^w_t, \mathbf{q}^w_{t+1} \in \mathbb{R}^{n_w}$ are well quantities from the training data, and $\hat{\mathbf{q}}^w_t, \hat{\mathbf{q}}^w_{t+1} \in \mathbb{R}^{n_w}$ are well quantities reconstructed (at time step $t$) and predicted (at time step $t + 1$) by the ROM. Recall that $n_w$ is the total number of wells. The variable $\gamma$ is a parameter that defines the weights for well-data loss in loss function $\mathcal{L}_p$.

The terms on the right hand side of Eq. 4.28 correspond to the flux and source terms in Eq. 4.1. In the examples in this chapter, we specify rates for injection wells and BHPs for production wells. With this specification, the loss on injection rates is zero. The key quantity to track for production wells is the well-block pressure for each well. This is because production rate is proportional to the difference between wellbore pressure (BHP in this case, which is specified) and well-block pressure. The proportionality coefficient is the product of phase mobility $\lambda_j$ and the Peaceman well index [89], which depends on permeability, block dimensions and wellbore radius. Because overall well rate in this case is largely impacted by well-block pressure, we set the well-block term on the right-hand side of Eq. 4.28 to $\gamma'\|\mathbf{p}^w_j - \hat{\mathbf{p}}^w_j\|_2$, where $\mathbf{p}^w_j \in \mathbb{R}^{n_p}$ and $\hat{\mathbf{p}}^w_j \in \mathbb{R}^{n_p}$ ($j = t, t + 1$) denote the true and ROM well-block pressures, and $n_p$ is the number of production wells. Here $\gamma'$ is a modified weight that accounts for the well index.

The physics-based loss function is computed by averaging $(\mathcal{L}_p)_i$ over all data points, i.e.,

$$
\mathcal{L}_p = \frac{1}{N_t} \sum_{i=1}^{N_t} (\mathcal{L}_p)_i.
$$

(4.29)

Combining the loss for data mismatch with this physics-based loss, the total loss
function becomes
\[ \mathcal{L} = \mathcal{L}_d + \alpha \mathcal{L}_p, \quad (4.30) \]

where \( \alpha \) is a weight term. Through limited numerical experimentation, we found \( \alpha = 0.033 \) and \( \gamma' = 20 \) to be appropriate values for these parameters. The E2C ROM prediction for the pressure field at a particular time, using the total loss function \( \mathcal{L} \), is shown in Fig. 4.5(d). Fig. 4.5(b) is again the high-fidelity solution (identical to that in Fig. 4.5(a)), and the difference map appears in Fig. 4.5(f). We see that the ROM prediction is noticeably improved when \( \mathcal{L}_p \) is included in the loss function. Specifically, the maximum pressure error is reduced from 97 psi to 16 psi, and the resulting field is smoother (and thus more physical). This demonstrates the benefit of incorporating physics-based losses into the E2C ROM.

### 4.2.6 E2C implementation and training details

To train the E2C model, we use a data set \( \mathcal{D} = \{(x_t, x_{t+1}, u_{t+1})_i\}, i = 1, \ldots, N_t, \) containing full-order states and corresponding well controls, where \( N_t \) is the total number of training run data points. In the examples in this chapter, we simulate a total of 300 training runs. This is many more than are used with POD-TPWL (where we typically simulate three or five training runs), but we expect a much higher degree of robustness with E2C. By this we mean that the ROM is expected to provide accurate results over a large range of control specifications, rather than over a limited range as in POD-TPWL.

Rather than train over all snapshots, here we set \( N_{\text{ctrl}} = N_{\text{tr}} = N_{\text{te}} = 20 \). This accelerates training and focuses ROM predictions on quantities of interest at time steps when the controls are changing. This results in a total number of data points of \( N_t = 300 \times 20 = 6000 \).

The gradient of the total loss function with respect to the model parameters \((\phi, \psi, \theta)\) is calculated via back-propagation through the embed-to-control framework. The adaptive moment estimation (ADAM) algorithm is used for this optimization, as it has been proven to be effective for optimizing deep neural networks [68]. The rate at which the model parameters are updated at each iteration is controlled by the learning rate \( l_r \). Here we set \( l_r = 10^{-4} \).
Normalization is an important data preprocessing step, and its appropriate application can improve both the learning process and output quality. For saturation we have $S \in [0, 1]$, so normalization is not required. Pressure and well data, including control variables, are normalized. Normalized rate $q^0$, and pressure (both grid-block pressure and BHP) $p^0$, are given by

$$q^0 = \frac{q - q_{\text{min}}}{q_{\text{max}} - q_{\text{min}}}, \quad p^0 = \frac{p - p_{\text{min}}}{p_{\text{max}} - p_{\text{min}}}.$$  (4.31)

Here $q$ denotes simulator rate output in units of $m^3$/day, $q_{\text{max}}$ and $q_{\text{min}}$ are the upper and lower injection-rate bounds, $p$ is either grid-block pressure or production-well BHP (units of psi), $p_{\text{min}}$ is the lower bound on BHP, and $p_{\text{max}}$ is 1.1 times the highest field pressure observed (the factor of 1.1 ensures essentially all data fall within the range).

Each full-order training simulation requires about 60 seconds to run on dual Intel Xeon ES-2670 CPUs (24 cores). Our E2C ROM is implemented using Keras [25] with TensorFlow [1] backend. The offline training process (excluding training simulation runtime) takes around 10-12 minutes on a Tesla V100 GPU node (exact timings depend on the memory allocated, which can vary from 8-12 GB). The model is applied on 100 test runs, which will be discussed in detail in the following section. Nearly all of the test results presented are based on the use of 300 training runs, though we also present summary error statistics using 100 and 200 training runs. Offline training for these case requires about the same amount of time as for 300 training runs, except for the direct savings in the full-order training simulations.

### 4.3 Results Using Embed-to-Control ROM

In this section, we describe the model setup for the oil-water simulations and we present simulation results for the deep-learning-based ROM. One of the test cases is considered in detail in this section; results for two additional test cases are provided in Appendix B. In this section we also present summary error results for all 100 test cases.
4.3. RESULTS USING EMBED-TO-CONTROL ROM

4.3.1 Model 4.1 setup: Gaussian reservoir

The geological model, in terms of the log-permeability field, is shown in Fig. 4.6. The locations of the four injection wells and five production wells are also displayed. The reservoir model contains 60 × 60 (total of 3600) grid blocks, with each block of dimensions 50 m × 50 m × 10 m. The correlation structure of the log-permeability field is characterized by an exponential variogram model, with maximum and minimum correlation lengths of ~1000 m and ~500 m, and an azimuth of 45°. The arithmetic mean permeability is 158 mD, and the standard deviation of log-permeability is 0.88. Permeability is taken to be isotropic, and porosity is set to a constant value of 0.2.

The relative permeability functions are given by

\[
k_{ro}(S_w) = k_{ro}^0 \left( \frac{1 - S_w - S_{or}}{1 - S_{wr} - S_{or}} \right)^a, \quad k_{rw}(S_w) = k_{rw}^0 \left( \frac{S_w - S_{wr}}{1 - S_{wr} - S_{or}} \right)^b,
\]

where \( k_{ro}^0 = 1.0, k_{rw}^0 = 0.7, S_{or} = 0.3, S_{wr} = 0.1, a = 3.6, \) and \( b = 1.5 \). Fluid densities are set to \( \rho_o = 800 \text{ kg/m}^3 \) and \( \rho_w = 1000 \text{ kg/m}^3 \), and viscosities are specified as \( \mu_o = 0.91 \text{ cp} \) and \( \mu_w = 0.31 \text{ cp} \). Capillary pressure effects are neglected.

![Figure 4.6: Model 4.1: log-permeability field and well locations](image)

The initial pressure at the top of the reservoir is 4712 psi (325 bar), and the initial water saturation is 0.1. The total number of primary variables in the system is 3600 × 2 = 7200. The model is run for a total of 2000 days. The injection wells
are controlled by specifying time-varying water rates, and the production wells are controlled by specifying time-varying BHPs. The controls for both production wells and injection wells are altered every 100 days, which means there are 20 control periods. Therefore, we have a total of \( 9 \times 20 = 180 \) control parameters over the entire simulation time frame. The range for the injection rates is between 1500 and 6500 bbl/day (between 238 and 1033 m\(^3\)/day). This is a very large range for well operation compared with what is often considered with ROMs [59, 60]. The range for production BHPs is 3770 to 3988 psi (between 260 and 275 bar).

The controls for the training and test runs are specified as follows. For each injection well, we randomly sample, from a uniform distribution between 2000 and 6000 bbl/day, a baseline injection rate \( q_{w_{\text{base}}} \). Then, at each control period, we sample uniformly a perturbation \( q_{w} \) over the range \([-500, 500]\) bbl/day. The rate for the control period is then prescribed to be \( q_{w_{\text{base}}} + q_{w} \). Producer BHPs at each control step are sampled uniformly over the range \([3770, 3988]\) psi. For production wells there is not a baseline BHP, and the settings from control step to control step are uncorrelated. This approach for specifying injection rates results in a wide range of solution behaviors (e.g., saturation distributions), since well-by-well injection varies considerably from run to run. This treatment also avoids the averaging effect that can occur if injection rates are not referenced to a baseline value \( q_{w_{\text{base}}} \). Well specifications for a test case, generated using this procedure, are shown in Fig. 4.7.

We perform 300 training simulations to construct the E2C ROM, except where otherwise indicated. The types of well schedules shown in Fig. 4.7 are intended to represent the well control profiles evaluated during optimization procedures, where the goal is to maximize oil production or profitability, or to minimize environmental impact or some measure of risk. We note finally that the dimension of the E2C latent space, \( l_z \), is set to 50.

### 4.3.2 Results for Test Case 1

In this section we present detailed results for a particular test case. These include well quantities (injection BHPs and production rates) and global quantities (pressure and saturation fields). The injection rate and BHP profiles for Test Case 1 are displayed in Fig. 4.7. Here we show the water rates for the four injection wells (Fig. 4.7(a)-(d)),...
and the BHPs for the five production wells (Fig. 4.7(e)-(i)).

We now assess the performance of the deep-learning-based ROM for this test case. The time-evolution of the global saturation field is first considered. Fig. 4.8 displays the saturation field at 200 days. In Fig. 4.8(a) the full-order saturation field (also referred to as the high-fidelity solution, HFS) is shown, and the corresponding E2C ROM result for saturation is presented in Fig. 4.8(b). The color scale indicates water saturation value (thus red denotes water). The close visual agreement between Fig. 4.8(a) and (b) suggests that the deep-learning-based ROM is able to provide accurate results for this quantity. The level of agreement between the two solutions is quantified in Fig. 4.8(c), where the difference between the HFS and ROM solutions is displayed. Note that the colorbar scale here is very different than that in Fig. 4.8(a) and (b). The error between the ROM and HFS results is clearly very small.

In order to better quantify the predictive ability of the E2C ROM, we introduce the concept of the ‘closest training run.’ We use this term to denote the specific training run, out of the 300 training runs performed, that most closely resembles (in a particular sense) the test case. The ‘distance’ between the test run and each of the training runs is quantified in terms of the Euclidean distance between their vectors of normalized control parameters, and the ‘closest training run’ ($k^*$) is the training run with the minimum distance. Specifically,

$$k^* = \arg \min_k \| U_{te}^0 - U_{tr,k}^0 \|_2^2,$$

where $k = 1, \ldots, 300$, denotes the index for the training runs, $U_{te} \in \mathbb{R}^{n_w \times N_{ctrl}}$ represents the control inputs for the test run, $U_{tr,k} \in \mathbb{R}^{n_w \times N_{ctrl}}$ indicates the control inputs for training run $k$, and the superscript 0 designates normalized pressures and rates in the controls, as per the normalizations in Eq. 4.31.

Eq. 4.33 provides a very approximate indicator of the ‘closest training run.’ This definition has the advantage of simplicity, though more involved (and computationally demanding) assessments would be expected to provide closer training solutions. These would, however, require the application of an approach along the lines of the point selection procedure used in POD-TPWL, as described in Section 3.1.4. This would entail computing a measure of distance (such as the $d(n, \tau, j)$ defined by Eq. 3.12 or
Figure 4.7: Test Case 1: well controls (Model 4.1)
4.3. RESULTS USING EMBED-TO-CONTROL ROM

Eq. 2.25) over many time steps for each training run. Since we have 300 training runs here (as opposed to three or five with POD-TPWL), this could become very time consuming. Thus we apply the simple approach defined in Eq. 4.33, with the recognition that more sophisticated (and presumably more accurate) procedures could be devised.

![Image](image_url)

(a) High-fidelity solution ($HFS_{test}$)  
(b) ROM solution ($ROM_{test}$)  
(c) $|HFS_{test} - ROM_{test}|$  
(d) $|HFS_{test} - HFS_{train}|$

Figure 4.8: Test Case 1: saturation field at 200 days (Model 4.1)

We now return to the global saturation results. Fig. 4.8(d) shows the difference between the ‘closest training run’ (determined as we just described and simulated at
high fidelity), and the test-case saturation fields. The colorbar scale is the same as in Fig. 4.8(c). The advantage of applying the deep-learning-based ROM is evident by comparing Fig. 4.8(c) and (d). More specifically, the error in Fig. 4.8(c) is about an order of magnitude less than the differences evident in Fig. 4.8(d).

![Saturation Field Comparison](image)

Figure 4.9: Test Case 1: saturation field at 1000 days (Model 4.1)

Figs. 4.9 and 4.10 display analogous results for saturation at 1000 and 1800 days. The evolution of the saturation field with time is apparent, and the deep-learning-based ROM solutions (Figs. 4.9(b) and 4.10(b)) are again seen to be in close visual
agreement with the HFS (Figs. 4.9(a) and 4.10(a)). The error maps in Figs. 4.9(c) and 4.10(c) further quantify the accuracy of the deep-learning-based ROM. These errors are quite small compared with the difference maps between the ‘closest training run’ and the HFS, shown in Figs. 4.9(d) and 4.10(d), which further illustrates the effectiveness of the ROM.

Note that in the ROM solutions, we do observe some local (unphysical) extrema within the saturation plumes. This is a minor issue here since the difference maps show small overall discrepancies between the ROM and high-fidelity solutions. In some cases, however, this could be a cause for concern. A potential remedy for this would be to add a term to the physics-based loss function such that local extrema in saturation that are inconsistent with the governing flow equations are penalized.

The global pressure field at particular times is also of interest. In Fig. 4.11(a) and (b) we display the HFS and ROM pressure solutions at 1000 days. The close visual agreement suggests that the deep-learning-based ROM is able to provide accurate (and smooth) pressure predictions. Fig. 4.11(c) shows the error map for the ROM solution, where we see that errors are indeed very small. These errors are much less than those for the ‘closest training run,’ which are shown in Fig. 4.11(d).

In many subsurface flow applications the well responses are of primary interest. E2C ROM predictions for these quantities will now be assessed. Since in this problem we specify injection rates and production well BHPs, the quantities of interest are injection well BHPs and oil and water production rates. Figs. 4.12 and 4.13 display the phase flow rates for Wells P1 and P2, which are the wells contributing most to total field production. Fig. 4.14 shows the BHP responses for all four injection wells. In all figures the black curves represent the full-order (reference) HFS, the red curves are the deep-learning-based ROM results, and the blue curves are the results for the ‘closest training run.’ A high degree of accuracy between the ROM and HFS results is consistently observed. The level of agreement in these essential quantities is enhanced through the additional weighting placed on well-block quantities in loss function $\mathcal{L}_p$ (see Eq. 4.28).

We present results for two more examples (Test Cases 2 and 3) in Appendix B. These results corroborate our observations here; namely, that the deep-learning-based ROM is able to accurately predict both global saturation and pressure distributions.
(a) High-fidelity solution ($\text{HFS}_{\text{test}}$)

(b) ROM solution ($\text{ROM}_{\text{test}}$)

(c) $|\text{HFS}_{\text{test}} - \text{ROM}_{\text{test}}|$  

(d) $|\text{HFS}_{\text{test}} - \text{HFS}_{\text{train}}|$  

Figure 4.10: Test Case 1: saturation field at 1800 days (Model 4.1)
4.3. RESULTS USING EMBED-TO-CONTROL ROM

(a) High-fidelity solution (HFS\textsubscript{test})

(b) ROM solution (ROM\textsubscript{test})

(c) $|\text{HFS}_{\text{test}} - \text{ROM}_{\text{test}}|$

(d) $|\text{HFS}_{\text{test}} - \text{HFS}_{\text{train}}|$

Figure 4.11: Test Case 1: pressure field at 1000 days (all colorbars in units of psi, Model 4.1)
CHAPTER 4. DEEP-LEARNING-BASED ROM

Figure 4.12: Test Case 1: production rates for Well P1

Figure 4.13: Test Case 1: production rates for Well P2 (Model 4.1)
4.3. RESULTS USING EMBED-TO-CONTROL ROM

(a) Well I1

(b) Well I2

(c) Well I3

(d) Well I4

Figure 4.14: Test Case 1: injection BHPs (Model 4.1)
and well quantities of interest.

Finally, we discuss the timings for the high-fidelity and ROM runs. The high-fidelity test cases take 60 seconds each to simulate using AD-GPRS on a node with dual Intel Xeon CPUs (24 cores). The full batch of 100 test cases can be evaluated using the E2C ROM in about 2 seconds on a Tesla V100 GPU node with about 1 GB of memory allocated. A direct (though simplistic) comparison indicates a speedup factor of 3000.

### 4.3.3 Results and error measurement for all test cases

In this section we assess the accuracy of the ROM results for the full ensemble of 100 test cases. We first consider field cumulative oil and water production, which are given by

$$Q_j = \int_0^T \sum_{w=1}^{n_p} q_{jw}(t) \, dt.$$  \hspace{1cm} (4.34)

Here $j = o, w$ denotes the phase, $n_p$ is the total number of production wells, $T$ designates the total simulation time, and $q_{jw}(t)$ represents the fluid rate for phase $j$ at time step $t$.

In Fig. 4.15 we present crossplots of $Q_o$ and $Q_w$ for the HFS and ROM solutions for the 100 test cases. The three $\times$’s on each plot indicate the results for Test Cases 1, 2 and 3. It is evident that these cases are quite different in terms of $Q_o$ and $Q_w$, and in this sense span the range of the 100 test cases. We see that the points in both plots fall near the 45° line, which demonstrates that our ROM solutions are in close agreement with the HFS. The results for $Q_w$ in Fig. 4.15(b) indicate that the ROM consistently under-predicts cumulative water production. The under-prediction is relatively small, however, as the range covered in this plot is narrow. Note finally that a slight over-prediction for cumulative oil production is evident in Fig. 4.15(a).

We now introduce a number of error measures, which will be used to assess the general performance of the E2C ROM. These error metrics closely follow those used in Section 2.3, though we present them again here for completeness. The relative
4.3. RESULTS USING EMBED-TO-CONTROL ROM

(a) Cumulative oil production

(b) Cumulative water production

Figure 4.15: Cumulative oil and water production for all 100 test cases (Model 4.1)

error for oil or water production rate, for a single production well $p$, is defined as:

$$
E_{p,j} = \frac{\int_0^T |q_{\text{ROM}}^{j,p}(t) - q_{\text{HFS}}^{j,p}(t)| \, dt}{\int_0^T |q_{\text{HFS}}^{j,p}(t)| \, dt},
$$

where $j = o, w$ is the fluid phase, $q^{j,p}(t)$ is the oil or water production rate at time $t$ for production well $p$, the subscripts HFS and ROM denote the high-fidelity and ROM results, and $T$ is the total simulation time. We define the error for overall production rate, $E_r$, in terms of $e_o$ and $e_w$ for all production wells, as:

$$
E_r = \frac{1}{n_p} \sum_{p=1}^{n_p} (e_{p,o}^p + e_{p,w}^p),
$$

where $n_p$ is the total number of production wells. Similarly, the relative error in injection BHP for a single injection well $i$ is defined as:

$$
e_{\text{BHP}}^i = \frac{\int_0^T |p_{\text{ROM}}^{w,i}(t) - p_{\text{HFS}}^{w,i}(t)| \, dt}{\int_0^T |p_{\text{HFS}}^{w,i}(t)| \, dt},
$$

where $p^{w,i}(t)$ denotes the injection BHP at time $t$ for injection well $i$. The overall
injection well BHP error $E_{\text{BHP}}$ is then given by:

$$
E_{\text{BHP}} = \frac{1}{n_i} \sum_{i=1}^{n_i} e_i^{\text{BHP}},
$$

(4.38)

where $n_i$ is the total number of injection wells.

Error in global quantities is also of interest. We define global pressure and saturation error as:

$$
E_v = \sum_{k=1}^{n_b} \int_0^T |v^k_{\text{ROM}} - v^k_{\text{HFS}}| dt, \quad (4.39)
$$

where $v^k$ denotes the global variable of interest in grid block $k$ (pressure $p^k$ or saturation $S^k$), and $n_b$ is the total number of grid blocks in the model.

These four error quantities are displayed as the red points in Fig. 4.16. We also evaluate these errors for the ‘closest training run’ for all test cases. In the plots, the points are ordered by increasing error for the ‘closest training run’ (blue points). Results for Test Cases 1, 2 and 3 are indicated in each plot. We see that the ROM errors are consistently very small, while the errors for the ‘closest training run’ are large in many cases. Interestingly, the ROM errors do not appear to depend on the error associated with the ‘closest training run.’ This is a desirable feature as it suggests a high degree of robustness in the E2C ROM.

We now briefly consider the use of smaller numbers of training runs in the construction of the E2C ROM. For these cases we present only summary error results. Fig. 4.17 displays the four relative errors considered above, in terms of box plots, for 100, 200 and 300 training runs. In each box, the central orange line indicates the median error, and the bottom and top edges of the box show the 25th and 75th percentile errors. The ‘whiskers’ extending out from the boxes indicate the minimum and maximum errors. There is significant improvement in ROM accuracy as we proceed from 200 to 300 training runs. In future work it will be useful to establish approaches to determine the required number of training runs.

Because it is difficult to display the errors for Test Cases 1, 2 and 3 in the box plots in Fig. 4.17, we present them in Table 4.1. These results are with 300 training runs. Note that the average values for $E_r$, $E_{\text{BHP}}$, $E_S$ and $E_p$ across all 100 test cases are about 0.14, 0.02, 0.04 and 0.002, respectively. The error values for the three test
4.3. RESULTS USING EMBED-TO-CONTROL ROM

(a) Production rates \( (E_r) \)

(b) Injection BHPs \( (E_{\text{BHP}}) \)

(c) Saturation field \( (E_S) \)

(d) Pressure field \( (E_p) \)

Figure 4.16: Errors for quantities for interest (Model 4.1)
(a) Error in production rates ($E_r$)

(b) Error in injection BHPs ($E_{BHP}$)

(c) Error in saturation field ($E_S$)

(d) Error in pressure field ($E_p$)

Figure 4.17: ROM error with different numbers of training runs (Model 4.1)
cases shown in the table can be seen to represent a reasonable spread among the full set of test cases. It is of interest to observe that the four errors do not appear to be closely correlated within a particular test case. For example, in Test Case 1, $E_p$ is in the 98th percentile, while $E_{BHP}$ is in the 15th percentile.

<table>
<thead>
<tr>
<th>Case</th>
<th>Error</th>
<th>$E_{BHP}$</th>
<th>$E_S$</th>
<th>$E_p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case 1</td>
<td>Error</td>
<td>0.11</td>
<td>0.0054</td>
<td>0.0418</td>
</tr>
<tr>
<td></td>
<td>Percentile</td>
<td>37</td>
<td>15</td>
<td>70</td>
</tr>
<tr>
<td>Test Case 2</td>
<td>Error</td>
<td>0.19</td>
<td>0.0064</td>
<td>0.0501</td>
</tr>
<tr>
<td></td>
<td>Percentile</td>
<td>96</td>
<td>29</td>
<td>99</td>
</tr>
<tr>
<td>Test Case 3</td>
<td>Error</td>
<td>0.16</td>
<td>0.012</td>
<td>0.0423</td>
</tr>
<tr>
<td></td>
<td>Percentile</td>
<td>77</td>
<td>84</td>
<td>73</td>
</tr>
</tbody>
</table>

Table 4.1: Errors and percentiles for test cases (Model 4.1)

4.4 Summary

In this chapter, we introduced a deep-learning-based reduced-order modeling procedure for oil-water reservoir simulation. The method was adapted from the existing embed-to-control (E2C) procedure, though some key modifications were introduced. Essentially, the ROM consists of an auto-encoder (AE) and a linear transition model. An additional physics-based loss function was combined with the data-mismatch loss function to enhance consistency with the governing flow equations. We showed that the various E2C ROM steps are very analogous to those used in (physics/numerics-based) POD-TPWL. Online (runtime) speedups of $O(1000)$, for the E2C ROM relative to AD-GPRS full-order simulations, were observed for the case considered. ROM accuracy was assessed for key quantities of interest, including well injection BHPs, oil and water production rates, and global pressure and saturation fields. The E2C ROM was shown to be consistently accurate over the full set of test runs.
Chapter 5

Conclusions and Future Work

5.1 Summary and Conclusions

The main goals of this dissertation were to develop and apply POD-TPWL reduced-order modeling (ROM) procedures for coupled flow-geomechanics problems and carbon storage operations, and to introduce a deep-learning-based ROM, embed-to-control (E2C), for subsurface flow simulation. The POD-TPWL ROM procedure entails projection to a low-dimensional subspace (using a basis matrix derived from the proper orthogonal decomposition of training state snapshots) in combination with trajectory piecewise linearization (in which new solutions are represented as linearizations around training solutions). The E2C ROM consists of an auto-encoder (AE) to project the system to a low-dimensional subspace and a linear transition model to approximate the system dynamics in low dimension. The construction of either the POD-TPWL or the E2C ROM requires offline computations, which include performing training runs (3-5 training runs for POD-TPWL, and 100-300 training runs for E2C) and preprocessing steps. However, the runtime speedups for both procedures are significant – for the cases considered here, we achieved $O(100)$ speedup for POD-TPWL and $O(1000)$ for E2C. Given the offline costs and online speedups, the use of these ROMs is only appropriate when many related runs are to be performed, which is the case in production optimization computations, data assimilation and uncertainty assessments.

In the first part of this dissertation, the POD-TPWL procedure was extended
to handle coupled flow-geomechanics problems. To treat the geomechanical equilibrium equations and the associated displacement variables, along with the oil-water flow equations and variables, a comprehensive extension of the general POD-TPWL methodology was implemented. This POD-TPWL formulation included the use of least-squares Petrov-Galerkin projection for constraint reduction and multiple derivatives for linearization. Generally accurate POD-TPWL results for key quantities of interest, including well injection and production rates, well failure criterion, and global pressure, saturation and principal stress fields, were achieved. In our evaluations, we performed either three or five full-order training runs. The offline POD-TPWL model construction required the equivalent of five or eight full-order simulations, depending on the number of training runs used.

The coupled flow-geomechanics POD-TPWL model was tested extensively on a 2D case with about 19,000 grid blocks and was also applied to a prototype 3D example with around 15,000 grid blocks. For the 2D system, we studied the impact of the training procedure and the level of perturbation (in test-case BHP profiles relative to those used in the training runs) on POD-TPWL error for key quantities of interest. A total of 20 test simulations were run for each perturbation level and each training procedure. Error was found to increase systematically with perturbation level, to decrease with the number of training runs performed, and to decrease when a degree of randomness was incorporated into the training-run BHP schedules.

In the second part of this dissertation, the POD-TPWL methodology was applied for the simulation of the injection stage of a carbon storage operation. This work represents an extension of the author’s Master’s research [58]. New POD-TPWL features introduced here include the use of multiple derivatives (i.e., multiple solutions around which linearization is performed), and a focus on state variables at particular locations in the domain rather than just well variables. The latter issue arises because we are concerned with the location of the CO$_2$ plume, in contrast to, e.g., oil field problems where the quantities of interest are often well injection and production rates. We tested the new POD-TPWL treatments on two examples, including a 3D aquifer characterized by a Gaussian log-permeability field with around 36,000 grid blocks, and a 3D channelized aquifer system with about 15,000 grid blocks. For both cases, we demonstrated that the use of multiple derivatives is capable of improving
POD-TPWL accuracy for key quantities such as well BHPs and CO$_2$ molar fractions in particular regions (i.e., plume location). Overhead (preprocessing) computation equivalent to about 6.7 full-order simulations was required to form the POD-TPWL model when five training runs were used.

We then applied this POD-TPWL formulation to an optimization problem, where the objective was to minimize the total CO$_2$ at a target layer at the end of the injection period. The control variables were the time-varying well injection rates, with a constraint that a prescribed total (field) CO$_2$ injection rate be met at all times. A mesh adaptive direct search (MADS) algorithm was applied for this optimization. Because POD-TPWL-based optimization is extremely fast, multiple optimization runs (starting from different initial guesses) can be performed efficiently. For the channelized system we achieved a reduction of about 70% in the total CO$_2$ in the target layer (relative to the initial-guess result). This reduction is essentially the same as that observed using optimization based on high-fidelity simulations. Thus the efficacy of POD-TPWL-based optimization was clearly illustrated.

In the third part of this dissertation, we introduced a deep-learning-based ROM. The procedure was adapted from the existing embed-to-control (E2C) formulation in [113]. The E2C ROM includes an auto-encoder and a linear transition model. We also incorporated a physics-based loss function, which was shown to provide improved pressure solutions. Although it is based on deep-learning concepts and methods, the various E2C ROM steps were shown to be very analogous to those used in POD-TPWL. In most of our evaluations, we performed 300 training runs in the offline step. Excluding the run time for the training simulations, the offline model construction required 10-12 minutes for ROM training using a Tesla V100 GPU.

The E2C ROM was tested on 2D oil-water reservoir simulation problems involving a heterogeneous permeability field with 3600 grid blocks. Large variations (relative to training runs) in injection and production well settings were prescribed in the test cases. A total of 100 test cases were considered. ROM accuracy was assessed for key quantities of interest, including well injection BHPs, phase production rates, and global pressure and saturation fields. The E2C ROM was shown to be consistently accurate over the full set of test runs. ROM error was seen to be much lower than that for the ‘closest training run’ (appropriately defined). Error was found to increase,
5.2. RECOMMENDATIONS FOR FUTURE WORK

However, if 100 or 200 training runs were used instead of 300. Note that the number of training runs used with the E2C ROM is much more than were used with POD-TPWL (in the two applications considered in this work, we used at most five training runs). As a result, the E2C ROM is able to provide predictions over a significantly larger range of well control settings.

5.2 Recommendations for Future Work

- For the POD-TPWL procedure for coupled flow-geomechanics problems, the current 3D model is still a prototype, and we found that the robust linkage between AD-GPRS and POD-TPWL (in terms of the behavior of the derivative matrices output by AD-GPRS and required by POD-TPWL) was sensitive to the number of border cells included around the central reservoir region. Sensitivity to other problem specifications and parameters was also observed. We believe these issues are associated with the forward model itself, though this requires further investigation. Following resolution of this issue, POD-TPWL should be tested on field-scale 3D problems with more wells and realistic control settings.

- A limited assessment of our POD-TPWL model for a coupled flow-geomechanics system that included a fault suggested that the existing implementation is suitable for use in such cases. Future research should entail application of the method to investigate fault slip in coupled problems. In addition, once a full-order simulator that can model dynamic fracture propagation is available, POD-TPWL should be extended and tested for this class of problems.

- For CO₂ storage operations, POD-TPWL was shown to provide accurate solutions during the injection period of a sequestration operation. During this stage, the POD-TPWL solution is driven by the difference in the time-varying injection rates between the test run and the training runs. However, during the equilibration period (after injection has stopped), this driving force vanishes and our current procedure is not applicable since the perturbation on which it is based no longer exists. Thus the method must be extended to enable
its application during the equilibration period. We note also that, during this stage, additional effects such as heterogeneous capillary pressure and relative permeability hysteresis are important. Once these effects are included in the full-order training simulations, the resulting POD-TPWL model can be tested and enhanced as necessary.

- There are many other ways in which POD-TPWL could be enhanced for subsurface flow applications. These include extension of the method to treat more complicated multiphysics problems, and the linkage of POD-TPWL with other optimizers. Retraining strategies, in the context of optimization, should also be assessed and refined. Use of the method for uncertainty quantification and inverse modeling should be considered. Further testing to see if the (positive) impact of randomness in training is consistently observed, as was the case for the coupled flow-geomechanics study, should be performed.

- The E2C ROM should be extended to more complicated 3D problems and tested on realistic cases. Extension to 3D can be approached by replacing conv2D layers with conv3D layers. The possibility of applying E2C for unstructured grid systems should also be investigated. The E2C ROM can be readily used with various optimization algorithms for production optimization, and its performance in this setting should be evaluated. Importantly, the E2C ROM should be applicable for use with global as well as local optimization algorithms. This is in contrast to POD-TPWL (and other POD-based ROMs), which can only be expected to be accurate in more limited neighborhoods and are thus most suitable for local-search methods. It is also of interest to explore the potential of predicting flow responses with changing well locations. If this is successful, the ROM could be applied for well location optimization, or combined well location and control optimization problems.

- To improve the accuracy and robustness of the E2C framework for more challenging applications, more complicated encoder and decoder structures, such as denseNet [55], could be evaluated. In addition, combining E2C with a variational U-Net [29] for fast uncertainty quantification in reservoir simulation could also be of great value [62]. Finally, the auto-encoder used here could be
extended to a VAE or the uncertainty auto-encoder [44] to enable system and control uncertainties to be taken into account.
Appendix A

Embed-to-Control Network Architecture

The architecture of the encoder is summarized in Table A.1. The encoder has a stack of four encoding blocks, with 16, 32, 64 and 128 filters, respectively. The detailed structure of the encoding block is shown in Fig. A.1(a). The size of the filter refers to that for the conv2D layer within the encoding block. The dimensions for the batchNorm layer and the ReLU are consistent with the size of the output from the conv2D layer. The encoder also has a stack of three residual convolutional (resConv) blocks, each with 128 filters of size $3 \times 3 \times 128$ and stride 1. The filter size again refers to those for the conv2D layers within the resConv block. The detailed structure of the resConv block is shown in Fig. A.1(d). Note that before being fed into the dense layer, the output from the resConv block is reshaped from a 3D matrix of dimension $(N_x/4, N_y/4, 128)$ into a long vector. The dense layer at the end of the encoder can be treated as a linear transform with input size $N_x/4 \times N_y/4 \times 128$, and output size of latent dimension $l_z$ ($l_z = 50$ in the cases considered here). Here $N_x$ and $N_y$ are the height and width of the input images, which are both 60 in our cases.

The architecture of the decoder is summarized in Table A.2. The decoder structure is analogous to that of the encoder, but with the components in reversed order. The decoder is comprised of a dense layer, a stack of three resConv blocks, a stack of four decoding blocks, and a conv2D layer. Note that the output of the dense layer is a long vector of dimension $N_x/4 \times N_y/4 \times 128$, which is reshaped into a 3D matrix of size
(N_x/4, N_y/4, 128) before being fed into the resConv block. The detailed structure of the decoding block is shown in Fig. A.1(b).

<table>
<thead>
<tr>
<th>Layer</th>
<th>Filter number, size and stride</th>
<th>Output size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td></td>
<td>(N_x, N_y, 2)</td>
</tr>
<tr>
<td>Encoding block</td>
<td>16 of 3 \times 3 \times 2, stride 2</td>
<td>(N_x/2, N_y/2, 16)</td>
</tr>
<tr>
<td>Encoding block</td>
<td>32 of 3 \times 3 \times 16, stride 1</td>
<td>(N_x/2, N_y/2, 64)</td>
</tr>
<tr>
<td>Encoding block</td>
<td>64 of 3 \times 3 \times 32, stride 2</td>
<td>(N_x/4, N_y/4, 128)</td>
</tr>
<tr>
<td>Encoding block</td>
<td>128 of 3 \times 3 \times 64, stride 1</td>
<td>(N_x/4, N_y/4, 128)</td>
</tr>
<tr>
<td>ResConv block</td>
<td>128 of 3 \times 3 \times 128, stride 1</td>
<td>(N_x/4, N_y/4, 128)</td>
</tr>
<tr>
<td>ResConv block</td>
<td>128 of 3 \times 3 \times 128, stride 1</td>
<td>(N_x/4, N_y/4, 128)</td>
</tr>
<tr>
<td>ResConv block</td>
<td>128 of 3 \times 3 \times 128, stride 1</td>
<td>(N_x/4, N_y/4, 128)</td>
</tr>
<tr>
<td>Dense</td>
<td></td>
<td>(l_z, 1)</td>
</tr>
</tbody>
</table>

Table A.1: Network architecture for encoder

<table>
<thead>
<tr>
<th>Layer</th>
<th>Filter number, size and stride</th>
<th>Output size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td></td>
<td>(l_z, 1)</td>
</tr>
<tr>
<td>Dense</td>
<td></td>
<td>(N_x/4 \times N_y/4 \times 128, 1)</td>
</tr>
<tr>
<td>ResConv block</td>
<td>128 of 3 \times 3 \times 128, stride 1</td>
<td>(N_x/4, N_y/4, 128)</td>
</tr>
<tr>
<td>ResConv block</td>
<td>128 of 3 \times 3 \times 128, stride 1</td>
<td>(N_x/4, N_y/4, 128)</td>
</tr>
<tr>
<td>ResConv block</td>
<td>128 of 3 \times 3 \times 128, stride 1</td>
<td>(N_x/4, N_y/4, 128)</td>
</tr>
<tr>
<td>Decoding block</td>
<td>128 of 3 \times 3 \times 128, stride 1</td>
<td>(N_x/4, N_y/4, 128)</td>
</tr>
<tr>
<td>Decoding block</td>
<td>64 of 3 \times 3 \times 128, stride 2</td>
<td>(N_x/2, N_y/2, 64)</td>
</tr>
<tr>
<td>Decoding block</td>
<td>32 of 3 \times 3 \times 64, stride 1</td>
<td>(N_x/2, N_y/2, 32)</td>
</tr>
<tr>
<td>Decoding block</td>
<td>16 of 3 \times 3 \times 32, stride 2</td>
<td>(N_x, N_y, 16)</td>
</tr>
<tr>
<td>Conv 2D</td>
<td>2 of 3 \times 3 \times 16, stride 1</td>
<td>(N_x, N_y, 2)</td>
</tr>
</tbody>
</table>

Table A.2: Network architecture for decoder

Fig. A.1 shows the detailed structure for the encoding blocks, the decoding blocks, the transformation blocks and the resConv blocks. An encoding block is a sequential combination of a 2D convolutional layer (conv2D), a batch-normalization (bathNorm) layer, and a rectified linear unit (ReLU). A reduction of spatial dimension (down-sampling) is achieved through tuning stride size for the conv2D layer (e.g., stride size of two for the encoding blocks in Table A.1). The decoding block has a 2D unpooling
APPENDIX A. EMBED-TO-CONTROL NETWORK ARCHITECTURE

layer, which increases the size of the input by repeating the rows and columns of the data, a 2D reflection padding layer, which also increases the size of input by padding the boundary on two of the dimensions of the data, a conv2D layer, a batchNorm layer, and a ReLU. The increase of spatial dimension (up-sampling) is achieved through the 2D unpooling and 2D reflection padding layers.

The structure of the resConv block closely resembles resNet, where an identity mapping is created to bypass the nonlinear layers. For a resConv layer, instead of the direct mapping of input $x$ to the target function $F(x)$, the nonlinear layer only needs to learn a residual mapping of $H(x) := F(x) - x$. This will be zero in the extreme (worst) case ($F(x) = x$) and guarantees that a deeper neural network will achieve higher (or at least equal) accuracy relative to its shallower counterpart. The nonlinear layer follows the standard structure of conv2D-batchNorm-ReLU. The transformation block has an architecture of dense-batchNorm-ReLU, where the dimension of the dense layer is set to 200 for the cases tested.

![Figure A.1: Blocks in embed-to-control models](image)

(a) Encoding block    (b) Decoding block    (c) Trans. block    (d) Res conv block

Figure A.1: Blocks in embed-to-control models
Appendix B

Additional Test Case Results

We now present E2C ROM results for Test Cases 2 and 3. The errors for quantities of interest for these cases are shown in Figs. 4.15 and 4.16 and in Table 4.1. Our descriptions here are brief since these results are very comparable to those discussed in Section 4.3.

B.1 Results for Test Case 2

Figs. B.1, B.2 and B.3 display the saturation field at 200, 1000 and 1800 days, and Fig. B.4 shows the pressure field at 1000 days, for this case. The evolution of saturation indicates a somewhat different sweep here compared to that of Test Case 1. Specifically, comparing Fig. B.3(b) to Fig. 4.10(b), we see that the water plume around Well I2 (upper right) is larger, and the water plumes around I1 (upper left) and I3 (lower left) are smaller in this case. Please refer to Fig. 4.6 for the exact well locations. The pressure map for Test Case 2 also shows a very different pattern compared to that of Test Case 1 at 1000 days. The difference maps shown in Figs. B.1(c), B.2(c), B.3(c) and B.4(c) indicate that the ROM predictions are again accurate for these global quantities.

Figs. B.5 and B.6 display oil and water production rates for Wells P1 and P2, which are the major contributors to total field production in Test Case 2. Fig. B.7 presents the injection BHPs for the four injection wells. The ROM solutions are again in close agreement with the high-fidelity solution, and significant discrepancy between
these solutions and the ‘closest training run’ is observed for most well quantities.

![Image](image-url)

(a) High-fidelity solution (HFS\text{test})  
(b) ROM solution (ROM\text{test})

(c) $|\text{HFS}_{\text{test}} - \text{ROM}_{\text{test}}|$  
(d) $|\text{HFS}_{\text{test}} - \text{HFS}_{\text{train}}|$

Figure B.1: Test Case 2: saturation field at 200 days (Model 4.1)

**B.2 Results for Test Case 3**

Analogous results for the evolution of saturation for Test Case 3 are shown in Figs. B.8, B.9 and B.10, and the pressure field at 1000 days is presented in Fig. B.11. The
B.2. RESULTS FOR TEST CASE 3

(a) High-fidelity solution \( (HFS_{test}) \)

(b) ROM solution \( (ROM_{test}) \)

(c) \( |HFS_{test} - ROM_{test}| \)

(d) \( |HFS_{test} - HFS_{train}| \)

Figure B.2: Test Case 2: saturation field at 1000 days (Model 4.1)
APPENDIX B. ADDITIONAL TEST CASE RESULTS

Figure B.3: Test Case 2: saturation field at 1800 days (Model 4.1)
B.2. RESULTS FOR TEST CASE 3

Figure B.4: Test Case 2: pressure field at 1000 days (all colorbars in units of psi, Model 4.1)
APPENDIX B. ADDITIONAL TEST CASE RESULTS

Figure B.5: Test Case 2: production rates for Well P1 (Model 4.1)

Figure B.6: Test Case 2: production rates for Well P2 (Model 4.1)
B.2. RESULTS FOR TEST CASE 3

(a) Well I1
(b) Well I2
(c) Well I3
(d) Well I4

Figure B.7: Test Case 2: injection BHPs (Model 4.1)
reservoir sweep is somewhat different here than in the other two cases, with the water plume around Well I4 (lower right) clearly smaller in this case. The difference maps again indicate a high level of accuracy for the E2C ROM. Production and injection well predictions appear in Figs. B.12, B.13 and B.14. We again obtain accurate ROM results for these important quantities.

Figure B.8: Test Case 3: saturation field at 200 days (Model 4.1)
B.2. RESULTS FOR TEST CASE 3

Figure B.9: Test Case 3: saturation field at 1000 days (Model 4.1)
APPENDIX B. ADDITIONAL TEST CASE RESULTS

(a) High-fidelity solution (HFS\textsubscript{test})

(b) ROM solution (ROM\textsubscript{test})

(c) |HFS\textsubscript{test} − ROM\textsubscript{test}|

(d) |HFS\textsubscript{test} − HFS\textsubscript{train}|

Figure B.10: Test Case 3: saturation field at 1800 days (Model 4.1)
B.2. RESULTS FOR TEST CASE 3

(a) High-fidelity solution (HFS\textsubscript{test})

(b) ROM solution (ROM\textsubscript{test})

(c) $|\text{HFS}_{\text{test}} - \text{ROM}_{\text{test}}|$

(d) $|\text{HFS}_{\text{test}} - \text{HFS}_{\text{train}}|$

Figure B.11: Test Case 3: pressure field at 1000 days (all colorbars in units of psi, Model 4.1)
APPENDIX B. ADDITIONAL TEST CASE RESULTS

(a) Oil rate

(b) Water rate

Figure B.12: Test Case 3: production rates for Well P1 (Model 4.1)

(a) Oil rate

(b) Water rate

Figure B.13: Test Case 3: production rates for Well P2 (Model 4.1)
B.2. RESULTS FOR TEST CASE 3

(a) Well I1

(b) Well I2

(c) Well I3

(d) Well I4

Figure B.14: Test Case 3: injection BHPs (Model 4.1)
Bibliography


