Optimizing Geological Carbon Storage Using Reinforcement Learning

by

Elijah French, Teo Nocita

A thesis submitted in conformity with the requirements
for the degree of

Department of Statistical Sciences
University of Toronto

Optimizing Geological Carbon Storage Using Reinforcement Learning

Elijah French, Teo Nocita

Department of Statistical Sciences
University of Toronto
2023

Abstract

With vast quantities of carbon being removed from the atmosphere there is an urgency to develop safe and effective strategies for its long-term storage. There are many methods of storing carbon but the most common and perhaps the most viable is geological carbon storage. This is where captured carbon is liquefied and injected deep underground. While this method is effective, it has the potential to cause serious adverse affects due to large increases in pressure. In this paper reinforcement learning is used to optimize geological carbon storage by finding a strategy that stores a sufficient amount of carbon while also keeping pressure levels low. To model the complex fluid dynamics underground a simulator called OPM Flow is utilized [5]. It is shown that reinforcement learning has the ability to find a strategy for storing carbon underground that outperforms a naive schedule and effectively manages pressure buildup.

Contents

Chapter 1

Introduction

In 2022, humans produced a total of 40.5 billion (short) tons of carbon dioxide (CO_2), an increase of about 0.9% from 2021 [1]. The Canadian government has committed to cut net greenhouse gas emissions to $40 - 45\%$ of 2005 levels before 2030 [14]. In order to meet this target, Canada will need to significantly reduce its net CO_2 emissions. Although large-scale reductions in industrial operations and consumption can help, some CO_2 emissions are necessary before a shift to a zero carbon economy is made. Capturing and then storing CO_2 may be the only way to reach this target.

There are various methods of storing CO_2 but perhaps the most scalable is geological carbon storage. Geological carbon storage is a method which involves firstly liquefying the captured CO_2 emissions and then injecting them deep underground into existing geological structures. Geological storage has various benefits over other carbon storage methods. In particular, it can make use of depleted oil and gas reservoirs and their existing equipment. It is estimated that the United States alone has the capacity to safely store about 3000 billion tons of CO_2 in existing geological formations [7], an enormous capacity when compared to annual global emissions. Further, the large amounts of CO_2 stored in this way typically has a comparatively small chance of leakage to the atmosphere in the long-term [3]. The Quest carbon storage project in Alberta, the largest geological carbon storage operation in Canada, stored just over 1.1 million tons of CO_2 in 2021 bringing its total to 7.4 million tons stored [10].

In the following report only operations that store CO_2 in deep saltwater (brine) aquifers will be considered. Such aquifers are estimated to have significantly more capacity globally than oil and gas reservoirs comparatively [2]. During storage it is important to monitor geophysical properties, in particular pressure levels. High levels of pressure can result in the fracturing of surrounding rock which may lead to liquid carbon or brine escaping into surrounding water tables. Such leakage can render fresh water in these systems non-potable. Another major concern associated with high pressure levels is induced seismicity. Induced seismic events from injection would likely not be directly dangerous to local populations, however, they could result in significant costs to surrounding infrastructure and housing [12].

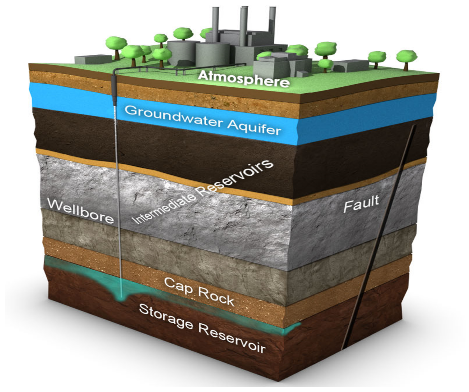

Figure 1.1: A Typical Geological Carbon Storage Operation

Storage operations generate revenue by injecting CO_2 and receiving carbon offset credits from the government. In addition to injection, the existing brine can be extracted. This is done in order to reduce pressure levels in the aquifers. Extracting and safely disposing this brine comes at a cost. Thus, rates of injection and extraction over time are controllable factors that can be used to optimize such operations. However, pressure dynamics in these environments are governed by complex multi-phase fluid flow differential equations, making optimal operation schedules difficult to solve analytically. Artificial Intelligence, in particular Reinforcement Learning, is used to learn these complex pressure dynamics and cost functions to optimize this process.

Chapter 2

Reinforcement Learning

Reinforcement Learning (RL) is a type of machine learning that aims to train a decision making agent to perform a task in a way that maximizes the chance of achieving some goal [13]. It does this by repeatedly attempting the task, taking actions, and learning from them. The agent will repeat actions it thinks are good whilst also randomly exploring other possibilities to find an optimal sequence of actions to take. This trade off between exploiting known actions and gaining knowledge on new actions is a key component of RL.

The way a RL agent learns is through repeated interactions with its environment (see figure 2.1). At time t, the agent observes the current state s_t and based off of this chooses some action a_t. The state then evolves into some new state s_{t+1} and the agent receives feedback in the form of a reward r_{t+1}. RL tasks are often episodic meaning that the agent tries to optimize the task over some finite time horizon T called an episode. The sets of possible states, actions, and rewards are denoted by $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$ respectively. Although RL environments can be incredibly complex, they are typically assumed to be Markov Decision Processes (MDPs). The way the environment evolves at time t can be characterized by the memoryless transition probabilities:

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

where $s, s' \in \mathcal{S}$, $a \in \mathcal{A}$, and $r \in \mathcal{R}$.

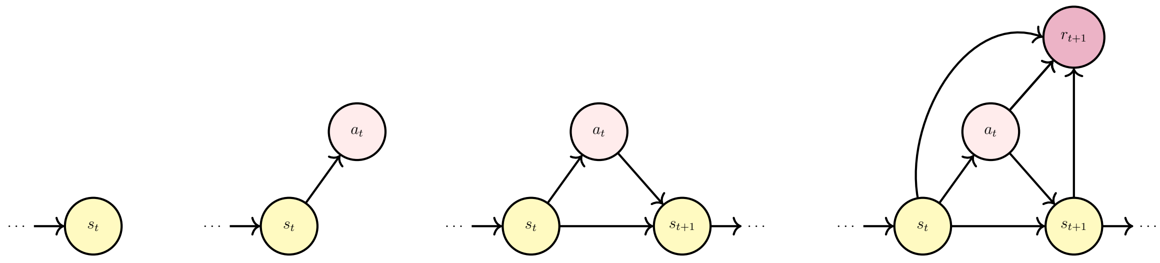

Figure 2.1: Evolution of the agent-environment interaction

An agent takes actions based on a function called a **policy** $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$. A policy is a mapping from states to distributions over actions where

$$\pi(a|s) = \mathbb{P}(A_t = a|S_t = s)$$

is the probability of choosing action a in state s. Note that degenerate distributions are possible in this framework allowing for agents to have deterministic policies.

The reward acts as a feedback signal for the agent that tells it how good or bad an action was. To formalize the RL problem, the agent wants to find a policy that maximizes its expected sum of discounted rewards:

$$\mathbb{E}_\pi \left[\sum_{t=0}^{T} \gamma^t R_{t+1} \middle| S_0 = s_0 \right] \tag{2.1}$$

where $\gamma \in [0, 1]$ is a discount factor. If $\gamma = 0$ the agent only cares about maximizing the immediate reward and if $\gamma = 1$ it wants to maximize the total reward it receives. The above expectation depends on the policy π since actions affect rewards directly and indirectly through state-evolution (see figure 2.1). In order to assess a policy, it is important to know the value of following it from a specific state. This is formalized by the **state-value** function:

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t} \gamma^k R_{t+k+1} \middle| S_t = s \right] \tag{2.2}$$

for $s \in \mathcal{S}$. This function gives the value of the state s at time t following the current policy π. Evaluating the quality of actions is also important. This is quantified by the **action-value** function

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a \right] \tag{2.3}$$

for $s \in \mathcal{S}$ and $a \in \mathcal{A}$. This is the value of taking action a in state s at time t and then following the policy π thereafter.

Letting $G_t = \sum_{k=0}^{T-t} \gamma^k R_{t+k+1}$ one can rewrite the state-value function as

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1}|S_t = s\right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(S_{t+1})|S_t = s\right] \end{aligned}$$

This equation is called the **Bellman equation**. It says that the value of a state s is the expected reward received in the state and the discounted value of the next state both under π. Similarly, one can derive the action-value Bellman equation:

$$\begin{aligned} Q_\pi(s, a) &= \mathbb{E}_\pi \left[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a\right] \\ &= \mathbb{E}_\pi \left[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a\right] \end{aligned}$$

As mentioned, the goal of an agent is to find the policy that maximizes equation (2.1). A slightly stronger condition for a policy is to maximize expected future reward from any state. Such a policy, denoted by π^*, is called optimal and satisfies the equation

$$\pi^* = \underset{\pi}{\operatorname{argmax}} V_\pi(s) \text{ for all } s \in \mathcal{S} \tag{2.4}$$

The notation is often changed for the optimal policy where $V_{\pi^*} = V^*$ and $Q_{\pi^*} = Q^*$. The optimal policy chooses the best action in each state and so the optimal policy's state-action value function satisfies

$$\begin{aligned} Q^*(s,a) &= \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma G_{t+1} | S_t = a, A_t = a\right] \\ &= \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma V^*(S_{t+1}) | S_t = a, A_t = a\right] \\ &= \mathbb{E}_{\pi^*}\left[R_{t+1} + \gamma \underset{a'}{\max} Q^*(S_{t+1}, a') \middle| S_t = a, A_t = a\right] \end{aligned}$$

Where the last equation follows from the fact that $V^*(S_{t+1}) = Q^*(S_{t+1}, \pi^*(S_{t+1})) = \max_{a'} Q(S_{t+1}, a')$ since π^* is optimal. This is referred to as the **Bellman optimality equation**. It says that under the optimal policy, the action-value function for state s and action a is equal to the expected reward plus the discounted action-value function in the next state when the best action is taken and the optimal policy is followed thereafter. The Bellman optimality equation is used in many RL algorithms.

Chapter 3

Deep Deterministic Policy Gradient (DDPG)

3.1 Background

In geological carbon storage, the possible actions (injection and extraction rates) form a bounded interval. The RL method chosen to optimize this process is Deep Deterministic Policy Gradient (DDPG), an actor-critic method made for continuous action spaces [4]. In training, a policy (actor) chooses actions and receives rewards. Its estimated action-value function (critic) is updated and then used to assess the quality of the actions chosen. The actor is subsequently updated, ensuring the policy approaches the optimal.

Before an in-depth explanation of these updates is made, a weaker case is considered. Assume that for a task the optimal policy is unknown but its action-value function $Q^*(s,a)$ is known and differentiable with respect to actions. Since the optimal policy maximizes its action-value function in each state s, it can be calculated from the equation $\pi^*(s) = \underset{a}{\text{argmax}} Q^*(s,a)$. However, due to the complexity of Q^* this maximum can often be computationally infeasible to find for each state encountered. Another approach is to use an artificial neural network (NN) and gradient ascent on its parameters to approximate π^* instead.

Firstly, a neural network for a policy (figure 3.1) is initialized and denoted by $\mu(\cdot|\theta) : \mathcal{S} \to \mathcal{A}$ where θ are its parameters. Secondly, initial learning and exploration rates $\alpha > 0$ and $\sigma > 0$ are chosen. Lastly, the agent repeats the task many times, encountering states s and taking actions $\mu(s|\theta) + \epsilon$ where $\epsilon \sim N(\vec{0}, \sigma^2\mathbf{I})$ is some noise that allows the agent to explore $\mathcal{S}$. After each attempt,

$$\nabla_\theta Q^*(s, \mu(s|\theta))$$

is calculated using the chain rule (Q^* is assumed differentiable with respect to its action argument) and the parameters are updated according to

$$\theta \leftarrow \theta + \alpha \nabla_\theta Q^*(s, \mu(s|\theta))$$

With a careful scheduling of α and σ, $\mu(\cdot|\theta)$ will approach π^*. In general, Q^* is unknown making the above process of approximating π^* impossible. DDPG fixes this issue by approximating Q^* with another NN, using it in the gradient computation above, and learning its parameters alongside the policy's.

3.2 Update Rules

The two NNs that define a policy and its action-value function are denoted by $\mu(\cdot|\theta^\mu)$ and $Q(\cdot,\cdot|\theta^Q)$ respectively (see figure 3.2). The DDPG algorithm ensures convergence of μ to the optimal policy through repeated parameter updates of Q and μ. To help with stability in these updates, two copies of the above (targets) are created $Q'(\cdot,\cdot|\theta^{Q'})$ and $\mu'(\cdot|\theta^{\mu'})$ where initially $\theta^{Q'} = \theta^Q$ and $\theta^{\mu'} = \theta^\mu$. A soft-update hyperparameter $\tau \in (0, 1]$ and learning rates $\beta, \alpha > 0$ are further initialized.

The task will be repeatedly attempted by $\mu(\cdot|\theta^\mu)$ and the experience tuples (s, a, r, s', d) are saved and learnt from. s' and d represent the new state and an indicator for whether the new state is terminal respectively.

3.2.1 Critic Update Rule

As mentioned in the introduction to RL, the optimal action-value function satisfies **Bellman optimality**:

$$Q^*(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a')|S_t = s, A_t = a] \quad \forall s \in \mathcal{S} \text{ and } a \in \mathcal{A}$$

This turns out to be a sufficient condition for optimality. Replacing Q^* with Q it can be used as a loss function that when minimized will ensure $Q \approx Q^*$. Finding the equation requires computing

$$\max_{a'} Q^*(S_{t+1}, a') = Q^*(S_{t+1}, \pi^*(S_{t+1}))$$

This is computationally infeasible to find for each state, however, since $\mu \approx \pi^*$:

$$\max_{a'} Q(S_{t+1}, a'|\theta^Q) \approx Q(S_{t+1}, \mu(S_{t+1}|\theta^\mu)|\theta^Q)$$

Using the target networks for stability in updates, the loss to minimize based off the experience tuple (s, a, r, s', d) is

$$\mathcal{L}_Q(\theta^Q) = \Big(Q(s,a|\theta^Q) - r - \gamma(1-d)Q'(s', \mu'(s|\theta^{\mu'})|\theta^{Q'})\Big)^2 \tag{3.1}$$

The $(1-d)$ is derived from the fact that the will be no future rewards if the next state is terminal. The gradient of $\mathcal{L}_Q$ is computed and the update

$$\theta^Q \leftarrow \theta^Q - \beta \nabla_{\theta^Q} \mathcal{L}_Q(\theta^Q)$$

is performed. The target network $Q'(\cdot,\cdot|\theta^{Q'})$ is then soft-updated according to

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

With proper scheduling of update hyper-parameters, these updates ensure that $\mathcal{L}_Q \to 0 \Rightarrow Q \to Q^*$

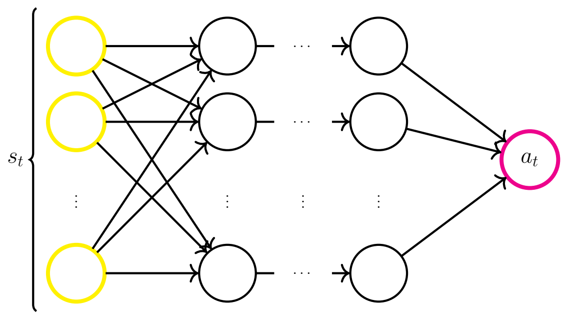

Figure 3.1: Actor NN Architecture

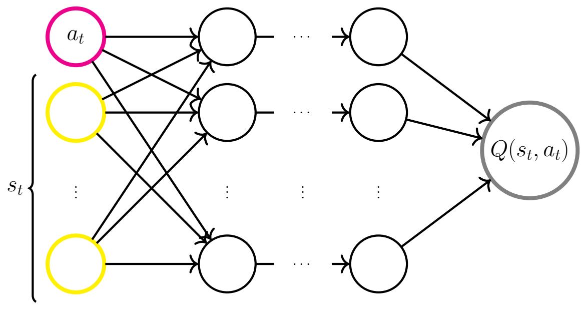

Figure 3.2: Critic NN Architecture

3.2.2 Actor Update Rule

If the policy is optimal, the agent will choose the action that maximizes its action-value function in every state. Therefore, in order for $\mu \to \pi^*$ $\mu(\cdot|\theta^\mu)$ the parameters θ^μ must be chosen to maximize

$$Q(s, \mu(s|\theta^\mu)|\theta^Q) \tag{3.2}$$

Using gradient ascent to ensure this results in the same parameter update mentioned above for the experience tuple (s, a, r, s', d):

$$\theta \leftarrow \theta + \alpha \nabla_{\theta^\mu} Q(s, \mu(s|\theta^\mu)|\theta^Q)$$

Since $Q \to Q^*$, this update will make μ approach π^*. The target actor network is then soft-updated accordingly

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

The critic (Q) guides the actor (μ) to make more optimal actions based on how it currently values the action taken by the actor. The actor and critic are used to update each other pushing themselves towards the optimal given enough exploration and iterations.

3.3 Algorithm

The DDPG algorithm begins by simulating an episode where actions are taken according to the randomly initialized actor network $\mu(\cdot|\theta^\mu)$. At each decision making time point, the agent stores the transitions (s, a, r, s', d) in a replay buffer $\mathbf{D}$. Once enough episodes are simulated and $\mathbf{D}$ is sufficiently large, N transitions $\{(s^i_{t_i}, a^i_{t_i}, r^i_{t_i}, s^i_{t_i+1}, d^i_{t_i})\}_{i \leq N}$ are sampled from $\mathbf{D}$ and means for equations 3.2 and 3.1 are calculated at which point gradient ascent and descent are respectively used to update the parameters of the actor and the critic. The purpose of the replay buffer $\mathbf{D}$ is to allow for transitions to be learned from multiple times in order to speed up convergence. This is especially useful when simulation of episodes is computationally expensive. This process is repeated, sampling from $\mathbf{D}$ and updating the parameters after each episode, until convergence is achieved. Once converged $\mu \approx \pi^*$.

Algorithm 1: Deep Deterministic Policy Gradient (Episodic) [4]

```
Initialize network parameters $\theta^Q$ and $\theta^\mu$ and an empty replay buffer $\mathbf{D}$. Initialize target
  networks setting $\theta^{Q'} = \theta^Q$ and $\theta^{\mu'} = \theta^\mu$. Set action exploration value $\sigma > 0$, soft-update
  value $0 < \tau < 1$, and sample mini-batch size $N$
for $i$ in iterations do
    Reset environment with initial state $s_0$
    while True do
        Take action $a_t = \mu(s_t|\theta^\mu) + \epsilon$ where $\epsilon \sim N(\vec{0}, \sigma^2\mathbf{I})$. Observe new state $s_{t+1}$, reward
          $r_t$, and terminal state signal $d_t$. Store $(s_t, a_t, r_t, s_{t+1}, d_t)$ in $\mathbf{D}$
        if $|\mathbf{D}| > N$ then
            Randomly sample a mini-batch of $N$ transitions $\{(s^i_{t_i}, a^i_{t_i}, r^i_{t_i}, s^i_{t_i+1}, d^i_{t_i})\}_{i \le N}$ from
              $\mathbf{D}$
            Calculate
                $y_i = r^i_{t_i} + \gamma(1 - d^i_{t_i})Q'(s^i_{t_i}, \mu'(s^i_{t_i+1}|\theta^{\mu'})|\theta^{Q'})$
            Update critic with gradient descent using
                $\nabla_{\theta^Q} \frac{1}{N} \sum_{i=1}^{N} (Q(s^i_{t_i}, a^i_{t_i}|\theta^Q) - y_i)^2$
            Update actor with gradient ascent using
                $\nabla_{\theta^\mu} \frac{1}{N} \sum_{i=1}^{N} Q(s^i_{t_i}, \mu(s^i_{t_i}|\theta^\mu))$
            Soft-update target networks
                $\theta^{Q'} \leftarrow \tau\theta^Q + (1 - \tau)\theta^{Q'}$
                $\theta^{\mu'} \leftarrow \tau\theta^\mu + (1 - \tau)\theta^{\mu'}$
        break if $d_t$
Take actions according to $\pi(s) = \mu(s|\theta^\mu)$
```

Chapter 4

Environment

4.1 OPM Flow

In order to optimize geological carbon storage it is necessary to know how pressure and saturation in underground saline aquifers evolves when carbon is injected or brine is extracted. OPM flow is a black oil simulator used to simulate fluid flow in porous rock (figure 4.1). The simulator has three components: water, gas, and oil each of which can take one of three phases: aqueous, oleic, or gaseous. OPM tracks phase changes and movements of these fluids. Furthermore, it is able to track solutions and precipitates that form from interactions between the fluids. The system is governed by the conservation of mass equations for the three components given by

$$\frac{\partial}{\partial t}(\phi_{ref} A_\alpha) + \nabla \mathbf{u}_\alpha + q_\alpha = 0$$

where α is either oil (o), gas (g), or water (w). A_α, $\mathbf{u}_\alpha$, and q_α are the accumulation term, flux, and well out flux density respectively for component α. ϕ_{ref} is a constant reference porosity for the reservoir. The fluid flow dynamics are also subject to Darcy's law and certain initial and boundary conditions.

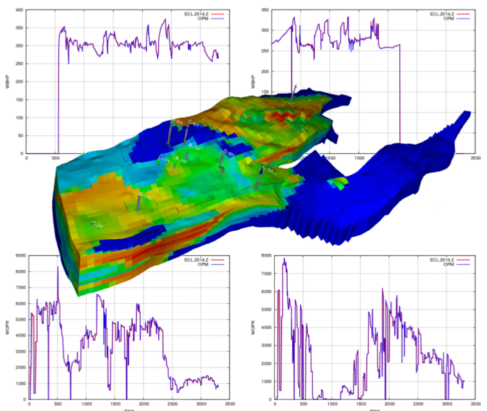

Figure 4.1: OPM simulating a non-rectangular reservoir

With OPM Flow, the reservoir is broken up into discrete cells. The equations governing fluid flow are discretized and solved implicitly. For the purposes of geological carbon storage, the main quantities of interest are carbon saturation and pressure. Using the discretized equations it is possible to obtain the carbon saturation and pressure for every cell in the reservoir as time evolves.

In order to model how fluids are injected or extracted into the reservoir iteslf, OPM uses a standard well model which describes the fluid composition inside the well. The total flow rate, fraction of water, and fraction of gas respectively are

$$\mathcal{Q}_t = \sum_{\alpha \in \{o,g,w\}} g_\alpha \mathcal{Q}_\alpha, \;\; F_w = \frac{g_w \mathcal{Q}_w}{\mathcal{Q}_t}, \;\; F_g = \frac{g_g \mathcal{Q}_g}{\mathcal{Q}_t}$$

where $\mathcal{Q}_\alpha$ is the flow rate and g_α is some weighting factor for component α.

4.2 CO2STORE Package

OPM Flow has the ability to model fluid dynamics for geological carbon storage in brine aquifers using the CO2STORE package. In this package the gas component is replaced by CO_2 and the oil component is replaced with brine. There is no water component. The reservoir is initially full of brine and CO_2 is injected which can stay separate from the brine or dissolve. Brine can also be extracted by wells in order to relieve pressure. When this package is active, properties such as density, viscosity, and enthalpy are computed internally as functions of pressure, temperature, and fluid composition using models from the relevant literature. Thermal properties of the fluids such as enthalpy not only depend on pressure, temperature, and dissolved CO_2, but also on the salinity of the brine which can be specified based on the region.

4.3 Reservoir Setup

A square reservoir (figure 6.1) is used to simulate storage. The reservoir is 7,500 feet underground, 18,000 feet across, and 50 feet vertically. In OPM this is discretized to a single-layered 60×60 grid. Although simulating a square reservoir may be restrictive, the porosity and permeability values of the rocks simulated are realistic. The pressure levels in the reservoir are initially in equilibrium and constant at 3500 PSI. In such reservoirs the chances of induced seismic events become increasingly more likely when the maximum pressure in the reservoir rises 1000 PSI above its initial level. Often times operations completely stop if this occurs. This will be reflected in the rewards described below. The interaction between the OPM simulator in the environment and the DDPG algorithm updates are demonstrated in figure 4.3. Every time step the agent observes the current state (or some restriction of the state), it chooses an amount to inject and extract until the next time period. OPM then evolves the reservoir based off of this, and some reward is received. After this, parameters are updated. In the following setups, the operation runs for a total of 24 years. Every two years the agent chooses an action based off of its current observation of the state.

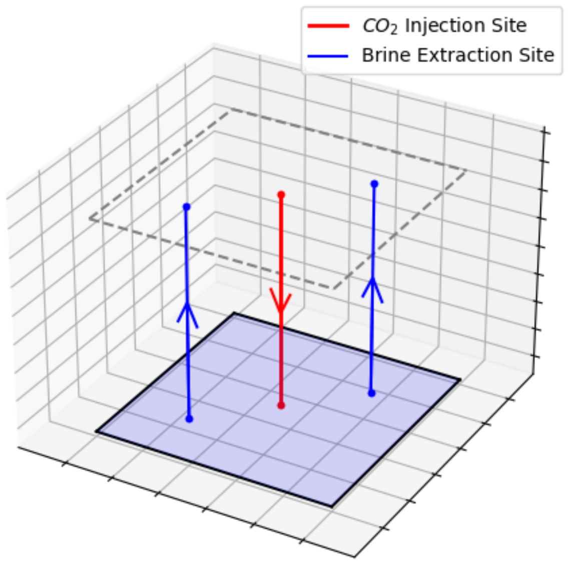

Figure 4.2: The reservoir used with injection and extraction sites

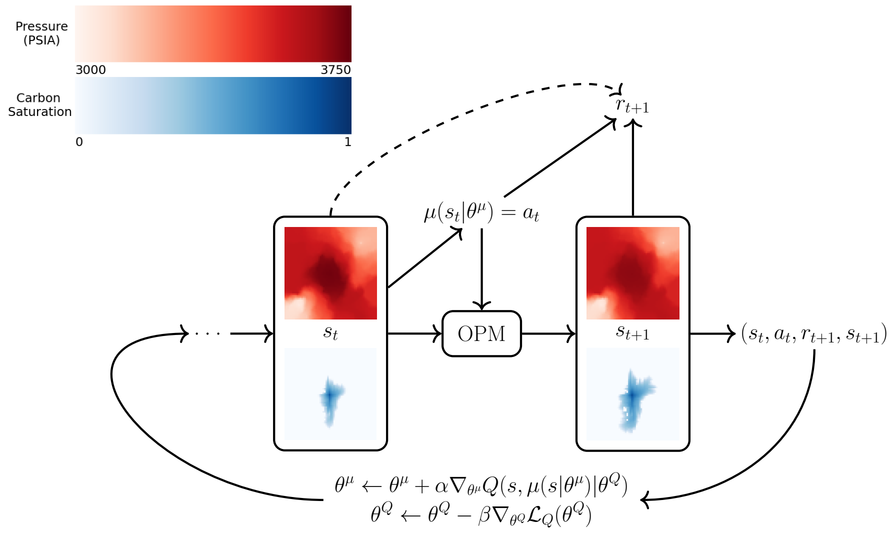
Pressure (PSIA)
3000
3750
Carbon Saturation
0
1
r_{t+1}
$\mu(s_t|\theta^\mu) = a_t$
s_t
OPM
s_{t+1}
$(s_t, a_t, r_{t+1}, s_{t+1})$
$\cdots$
$\theta^\mu \leftarrow \theta^\mu + \alpha\nabla_{\theta^\mu} Q(s, \mu(s|\theta^\mu)|\theta^Q)$
$\theta^Q \leftarrow \theta^Q - \beta\nabla_{\theta^Q}\mathcal{L}_Q(\theta^Q)$

Figure 4.3: The DDPG OPM interaction

Chapter 5

Framework I: Full Observability

5.1 States and Rewards

Under the full observability framework, it is assumed that at every decision making time step the agent observes the pressure and carbon saturation at every cell in the reservoir. This results in the state

$$s_t = (\mathbf{P}_t, \mathbf{S}_t)$$

where $\mathbf{P}_t$ and $\mathbf{S}_t$ are 60×60 arrays of pressure and carbon saturation values respectively. Because of the strong spacial correlations in the pressure and saturation grids, convolutional neural networks are used for both the actor and critic to efficiently process the states. Once the state is observed, the actor network will output a target injection rate for the agent to take for the next 2 year period

$$a_t = \nu_t$$

The maximum target rate that the agent can choose is 3.9 short tons per day. Since the rate chosen by the agent is a target, the actual amount of carbon injected in the period c_t may slightly differ from $\nu_t \Delta t$ but the difference is negligible. The agent chooses a new action every 2 years for a 24 year period.

As mentioned, in order to avoid significant chances of induced seismicity, it is best practice to shut down operations if the maximum pressure at any point in the reservoir rises 1000 PSI above the initial reservoir pressure. Further, there are often precautionary measures placed on lower non-terminal pressures. This is reflected in a stoplight type system for injection where green means it is safe to continue injecting, yellow indicates caution should be used, and red indicates the operations should stop. This is incorporated into the reward function for the agent with a sigmoid (5.2) which takes in the maximum pressure in the grid and outputs a penalty based on how much higher it is than the initial equilibrium pressure I. A sigmoid is used in order to smooth the rewards received by the agent to aid in learning. In addition to this soft penalty for pressures in the yellow zone, a harsh penalty is experienced when the maximum observed pressure rises above 1,000 PSI from its initial pressure. In this case the operation shuts down completely and no more rewards are received.

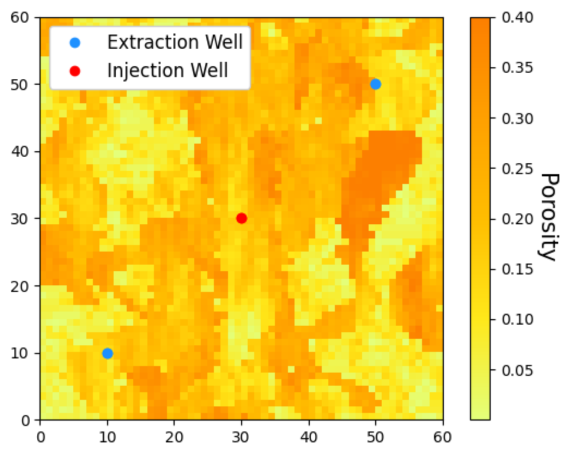

Figure 5.1: Full observability framework reservoir

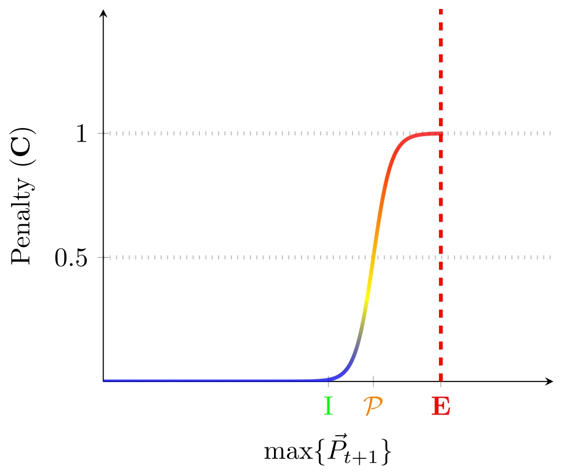

Figure 5.2: Penalty incurred for values of maximum pressure. Terminal pressure **E** is 1000 psi higher than I

Since the agent receives offset credits based on the amount of CO_2 injected, the goal is to inject as much CO_2 as possible while also not incurring any penalty associated with the pressure rising. If the maximum pressure is in the yellow zone, the agent will lose the full value of the injected carbon for the last injection period. This results in the following reward function:

$$r_t = \beta_\nu \nu_t (1 - \mathbf{C}(\max \vec{P}_t))$$

where C is the cost penalty from figure 5.2 and β_ν is the current financial benefit of injection.

5.2 Results

Due to the large state space in this framework, a substantial amount of computational power is needed to train this algorithm. As a result convergence was not acheived under this framework.

Chapter 6

Framework II: Partial Observability

6.1 States and Rewards

The second framework considered is one in which the relevant state factors are only partially observed. In geological storage operations, the pressure and saturation values in the entire reservoir are impossible to observe. However, injection and extraction wells often have the ability to detect pressure values. Further, observational specific wells can be placed in strategic locations to gain more information on how the reservoir is evolving. In order to emulate this restriction, another state observation setup is used. In particular, the agent is now restricted to only observing pressure values in 7 locations in the reservoir (including injection and extraction wells). This setup is depicted in figure 6.1. There are four observation wells, two extraction wells, and one injection well. The agent will be able to observe the pressure values in these seven locations and must make inferences on how pressure is evolving elsewhere in the grid.

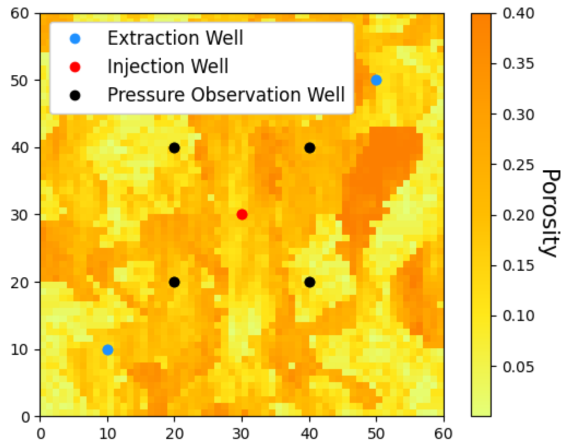

Figure 6.1: Partial observability framework reservoir

In addition to these pressure values, the agent knows how far into the 24 year 12 time-step episode they are. Thus, for the agent, the observed state of the reservoir at time t is

$$s_t = (t, P_{1,t}, ..., P_{7,t}) \tag{6.1}$$

In an extension to framework I, the agent now has control over extraction. Thus, every two years the agent must decide how much carbon to inject into the reservoir and how much brine to extract. So, the agent observes the state s_t and takes an action

$$a_t = (\nu_t, b_t) \tag{6.2}$$

Where ν_t and b_t represent injection and extraction rates respectively. Both extraction wells produce brine at the same rate chosen by the agent. ν_t and b_t can range to up to 0.4 and 1800 short tons respectively per day over the two year period.

Again, to be in accordance with the stoplight system, a max pressure close to $1,000$ psi above the initial value should result in no benefit for the agent from injection and a max pressure change above $1,000$ should end operations. Since brine extraction costs the firm, the reward function is defined to be

$$r_t = \beta_\nu \nu_t (1 - \mathbf{C}(\max \vec{P}_t)) - \beta_b b_t$$

where C is the cost penalty for pressure shown in figure 5.2. β_ν and β_b are constants that reflect the current financial benefit and cost to injection and extraction respectively. The reward constants chosen for this specification are $\beta_\nu = 2.7839$ and $\beta_b = 1.1403$. They reflect the current carbon offset price [10] and cost of brine production [8].

6.2 Results

The performance of the agent is compared with that of an optimal constant schedule found by trying all combinations of rates $\nu_t, b_t \in \{0, 0.1, 0.2, ..., 0.9, 1\}$ for every $t \leq T$. The optimal constant sequence and its respective cumulative normalized rewards are shown in figure 6.2. The optimal constant rates were found to be $\nu_t = 0.5$ and $b_t = 1$. This schedule allowed for the maximum pressure to remain below that of the high-pressure region. In the end, the optimal constant schedule received a total cumulative normalized reward of 11.2065.

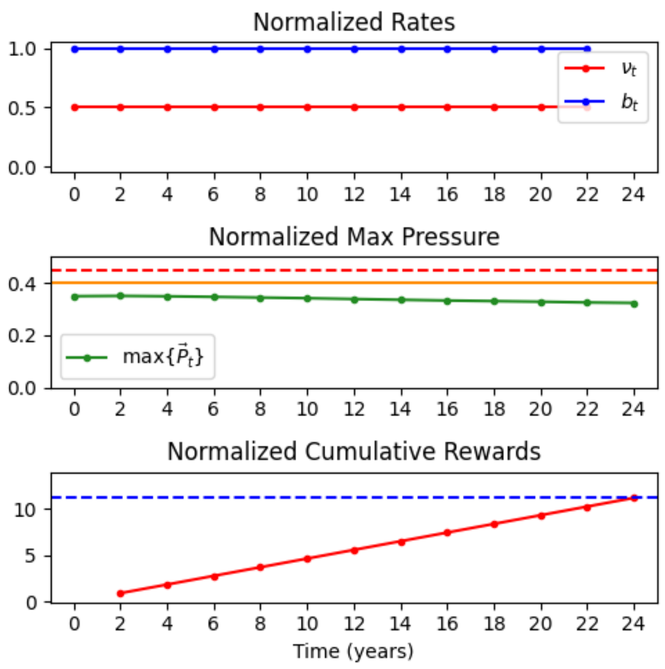

Figure 6.2: Optimal constant schedule with $\sum_{t=0}^{T} r_t = 11.2065$

After running the DDPG algorithm for 1000 iterations, the resultant schedule chosen was

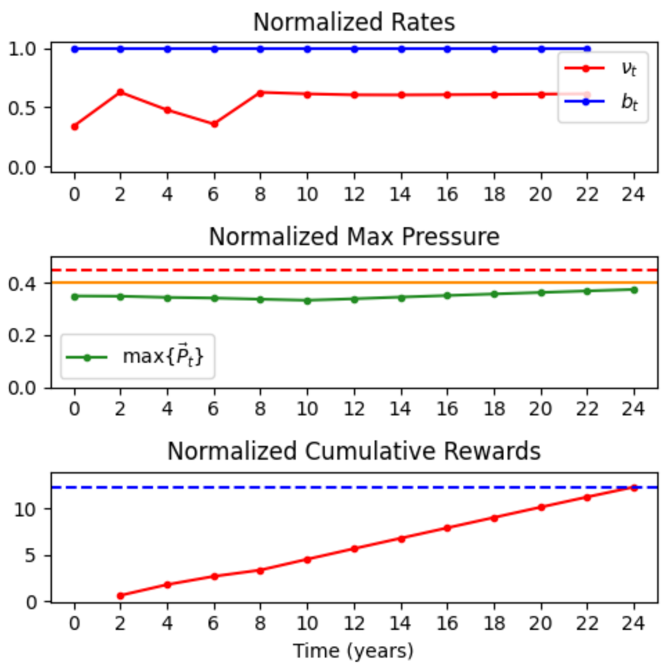

Figure 6.3: DDPG schedule after 1000 iterations with $\sum_{t=0}^{T} r_t = 12.3403$

Similarly to how the optimal constant schedule deals with pressure, the agent decides to maximize extraction throughout the episode. The optimal constant schedule has injection at 0.5 for the entire episode whilst the DDPG schedule deals with pressure build-up by fluctuating around a rate of 0.5 in the first half and then remaining relatively constant for the remainder of years. This schedule is able to maximize reward from injection by keeping the maximum pressure just below the high pressure level the entire time. This is in contrast to the constant schedule in which pressure is kept just below the high pressure threshold near the end of the episode. In the end, the DDPG agent after 1000 iterations is able to more efficiently control pressure levels in the reservoir and thus receives a higher total cumulative reward. The same model is trained for another 1000 iterations with a smaller exploration rate. The agent again decides to implement the maximum The resulting schedule increases cumulative rewards to 12.3723.

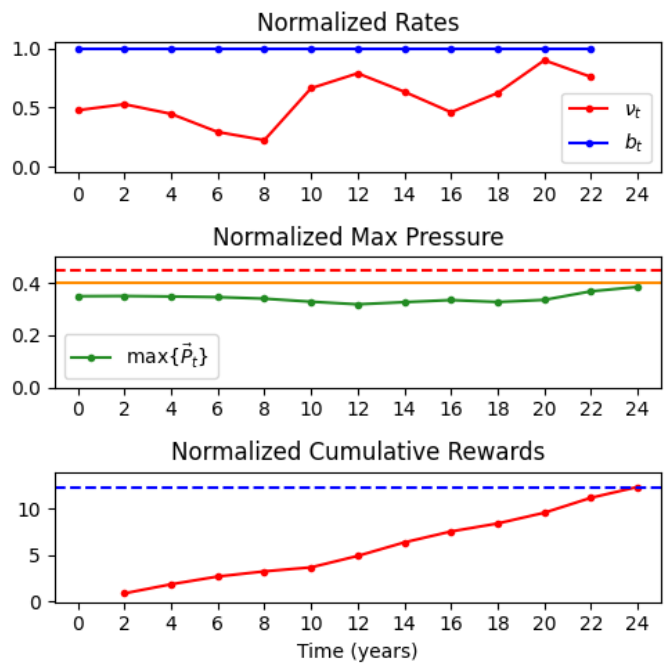

Figure 6.4: DDPG schedule after 1000 iterations with $\sum_{t=0}^{T} r_t = 12.3723$

This new schedule keeps a similar behaviour to the one above at the beginning of the episode in that it decides to decrease injection until year 6 and then increase thereafter (with a smaller initial injection rate). This allows the agent to remove pressure from the system and increase rates later on.

Chapter 7

Discussion and Conclusion

Artificial Intelligence has become a powerful tool to optimize operations in many industries. The above results show that carbon storage can also benefit from AI integration. The ability of the neural networks to learn these complicated pressure dynamics results in non-instinctive schedules of injection rates. This exemplifies the power of neural networks and the algorithms that apply them.

In the first, full observability specification, the state space and the convolutional neural networks used to observe it came at a high computational cost to train. This large parameter space combined with the low speed of OPM flow resulted in a lack of convergence for the DDPG model. With more computational time, convergence to an optimal would be likely. In the second, partial observability, specification the final schedule chosen by the agent uses a non-intuitive method to deal with pressure build-up. A fluctuation around the midpoint of injection rates allows for the model to gain rewards from high levels of CO_2 injection whilst keeping the pressure in the reservoir under control.

The above specifications imply that in a reservoir with realistic geophysical properties, an AI can learn to optimize scheduling. However, as with many machine learning algorithms, there is a boundary to the above agent's knowledge. If in the real world a state is experienced by the agent that it has not seen before, it may suggest an action which would be detrimental to operations. The suggestion for a carbon storage firm would then be to model their reservoir using geostatistical techniques, train an RL agent with realistic rewards, and use the resulting optimal schedule in conjunction with human expertise. Allowing for human intervention where necessary.

Various extensions can be made to this work, including training with realistically shaped reservoirs, introducing stochasticity to the offset credit price of carbon, and using geo-statistical inference to model reservoir characteristics. As mentioned, the OPM simulator has the ability to model multi-phase fluid flow in highly complicated geophysical structures. Moving to a model that has a geo-physically realistic shape would enhance results. The Canadian government's commitment to cut emissions has involved a scheduled increase of carbon offset prices overtime. Introducing a stochastic carbon injection price into the model would result in more realistic results. The cost of drilling wells for production or observation is tremendous. Often times 7 observation points (as used in the above model) is quite unrealistic. Using geo-physical observation methods and geo-statisical interpolation can cut down on such costs for firms and improve the implementation of such results.

In conclusion, the above findings show that reinforcement learning can be used in novel environments to find non-instinctive solutions to problems. These results should be investigated further with more realistic reservoir shapes, implementing full cost analyses, and stochasticity in prices. In addition to optimizing profit for a particular storage firm, this agent has shown efficiency in being able to store as much carbon as possible without inducing large amounts of pressure. If Canada is going to drastically reduce its net CO_2 emissions in the coming years, reinforcement learning can be a powerful tool to help do so.

Bibliography

[1] IEA (2023). *CO2 Emissions in 2022, IEA, Paris.* 2023. URL: https://www.iea.org/reports/co2-emissions-in-2022.

[2] Celia et al. *Status of CO2 storage in deep saline aquifers with emphasis on modeling approaches and practical simulations.* 2015. URL: https://doi.org/10.1002/2015WR017609.

[3] Kivi et al. *Multi-Layered Systems for Permanent Geologic Storage of CO2 at the Gigatonne Scale.* 2022. URL: https://doi.org/10.1029/2022GL100443.

[4] Lillicrap et al. *Continuous Control With Deep Reinforcement Learning.* 2016. URL: https://arxiv.org/abs/1509.02971.

[5] David Baxendale. *Open Porous Media OPM Flow Reference Manual.* 2023.

[6] *Deep Deterministic Policy Gradient.* 2018. URL: https://spinningup.openai.com/en/latest/algorithms/ddpg.html.

[7] *Geologic Sequestration of Carbon Dioxide: What You Should Know.* 2022. URL: https://ifsolutions.com/geologic-sequestration-of-carbon-dioxide-what-you-should-know/.

[8] Noh and Swidinisky. *Stochastic Control of Geological Carbon Storage Operations Using Geophysical Monitoring and Deep Reinforcement Learning.* 2023. URL: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4546728.

[9] Mathew Emmanuel Pineda. *Carbon Capture and Storage: Methods and Effectiveness.* 2019. URL: https://www.profolus.com/topics/carbon-capture-and-storage-methods-and-effectiveness/.

[10] Alberta Shell Canada Limited Calgary. *Quest Carbon Capture and Storage Project 2021 ANNUAL STATUS REPORT.* 2022. URL: https://open.alberta.ca/dataset/113f470b-7230-408b-a4f6-8e1917f4e608/resource/476a41bf-33a3-4f52-9436-1cff32f76eeb/download/quest-2021-annual-status-report-alberta-energy-regulator.pdf.

[11] David Silver et al. "Deterministic Policy Gradient Algorithms". In: *International Conference on Machine Learning.* 2014. URL: https://api.semanticscholar.org/CorpusID:13928442.

[12] Integrated Flow Solutions. *Geologic sequestration of carbon dioxide: What you should know.* 2022. URL: https://ifsolutions.com/geologic-sequestration-of-carbon-dioxide-what-you-should-know/.

[13] Barto Sutton R.S. *Reinforcement Learning: An Introduction.* 1998.

[14] John Paul Tasker. *Canada releases plan for a 40 per cent cut in carbon emissions by 2030*. 2022. URL: https://www.cbc.ca/news/politics/canada-2030-emissions-reduction-plan-1.6401228.

Appendix A

OPM Flow Dynamics

For each of the components in OPM the accumulation terms and fluxes are:

$$\begin{aligned} A_w &= m_\phi b_w s_w \\ A_o &= m_\phi (b_o s_o + r_{og} b_g s_g) \\ A_g &= m_\phi (b_g s_g + r_{go} b_o s_o) \end{aligned}$$

and

$$\begin{aligned} \mathbf{u}_w &= b_w \mathbf{v}_w \\ \mathbf{u}_o &= b_o \mathbf{v}_o + r_{og} b_g \mathbf{v}_g \\ \mathbf{u}_g &= b_g \mathbf{v}_g + r_{go} b_o \mathbf{v}_o \end{aligned}$$

The system is subject to the following closure equations and boundary conditions

$$\mathbf{v}_\alpha = -\lambda_\alpha \mathbf{K}(\nabla p_\alpha - \rho_\alpha \mathbf{g})$$

$$\begin{aligned} s_w + s_o + s_g &= 1 \\ p_{c,ow} &= p_o - p_w \\ p_{c,og} &= p_o - p_g \end{aligned}$$

Table A.1: Environment Variables

Variable	**Description**
ϕ	porosity
m_ϕ	pore volume multiplier as a function of pressure
ϕ_{ref}	constant reference porosity
p_α	pressure for phase α
b_α	shrinkage/expansion factor for phase α defined as ratio of surface volume at standard conditions to the reservoir volume for a given amount of fluid
r_{go}	gas-oil ratio of dissolved gas to oil in the oleic phase
r_{og}	oil-gas ratio of vaporized oil to gas in the gaseous phase
s_α	saturation of phase α
$p_{c,\alpha\beta}$	capillary pressure between phases α and β
$\mathbf{K}$	permeability of the porous medium
$k_{r;\alpha}$	relative permeability for phase α
μ_α	viscosity of phase α
λ_α	mobility of phase α given by ratio of relative permeability to viscosity
$\mathbf{v}_\alpha$	velocity of phase α
$\mathbf{u}_\alpha$	velocity of component α
$\rho_{S,\alpha}$	surface density of phase α at one atmosphere
ρ_α	density of phase α in the reservoir
$\mathbf{g}$	gravitational acceleration vector
q_α	well out flux density for pseudo component α

Let p_{bhp} be the bottom hole pressure which is the pressure measured at some reference point within the well. Volumetric flow rates are reservoir conditions are

$$q^r_{\alpha,j} = T_{w,j} M_{\alpha,j} \left[p_j - (p_{bhp,w} + h_{w,j})\right]$$

Table A.2: Well Model Variables

Variable	**Description**
$q^r_{\alpha,j}$	flow rate for phase α through connection j
$T_{w,j}$	connection transmissibility factor
$M_{\alpha,j}$	mobility for phase α at connection j
p_j	pressure in the grid block that contains connection j
$p_{bhp,w}$	bottom hole pressure of well w
$h_{w,j}$	pressure difference between well connection j and bhp depth

To keep the system closed the following conservation equation is introduced for each component

$$R_{\alpha,w} = \frac{A_{\alpha,w} - A^0_{\alpha,w}}{\delta t} + \mathcal{Q}_\alpha - \sum_{j \in C(w)} q_{\alpha,j} = 0$$

where $C(w)$ is the set of connections for well w, $q_{\alpha,j}$ is the flow rate of phase α through connection j under surface conditions, and $A_{\alpha,w}$ is the amount of component α in the well.

For wells controlled by bhp the governing equation is

$$R_{c,w} = p_{bhp,w} - p_{bhp,w}^{target} = 0$$

and for wells controlled by rate it is

$$R_{c,w} = \mathcal{Q}_{\alpha} - \mathcal{Q}_{\alpha}^{target} = 0$$

where $p_{bhp,w}^{target}$ is the desired bhp for the well and $\mathcal{Q}_{\alpha}^{target}$ is the desired surface volume rate for component α. For more information on fluid flow dynamics including the salt precipitation model see the OPM manual.

Table A.3: Framework I Hyperparameters

Hyperparameter	**Value**
$\mu(\cdot\|\theta^{\mu})$ (Actor) layers	5

Table A.4: Framework II Hyperparameters

Hyperparameter	**Value**
$\mu(\cdot\|\theta^{\mu})$ (Actor) layers	5
$\mu(\cdot\|\theta^{\mu})$ nodes	40 for each layer
$Q(\cdot\|\theta^{Q})$ (Critic) layers	5
$Q(\cdot\|\theta^{Q})$ nodes	40 for each layer
$Q(\cdot\|\theta^{Q})$ nodes	40 for each layer
$Q(\cdot\|\theta^{Q})$ nodes	40 for each layer