ABSTRACT

The severe global warming issues have driven the energy industry toward renewable resources and low-carbon emissions activities. In response, the petroleum sector has initiated several projects to store carbon dioxide (CO_2) in underground reservoirs. This thesis utilizes available open-source software tools to model and optimize CO_2 storage. Specifically, the project is based on two software programs: *Flow*, a reservoir simulator that includes options to simulate CO_2 flow behavior in the reservoir, and *FieldOpt*, a field development optimization framework that optimizes engineering and geological parameters of reservoir simulation. This thesis aims to extend the *FieldOpt* software package to improve capabilities related to CO_2 injection projects.

This thesis will consider the following topics related to modeling of CO_2 storage: simulate fluid flow in the reservoir, analyze the post-simulation data, update the simulation case according to optimization algorithms, and re-simulate the model. Running *FieldOpt* would integrate all the processes above, using the simulator *Flow* for the reservoir simulation, and in an interactive optimization process finding a local optimum.

This project will extend the *FieldOpt* software to enable options necessary for the optimization of CO_2 storage. The *FieldOpt* software is written in the object-oriented programming language C++. This thesis will therefore discuss variables, functions, classes, and modules deployed in the *FieldOpt* software package that are relevant for extensions towards CO_2 storage. While extending the software by developing new features, current functionality should be kept intact, and the overall structure of the software source code will be maintained. As a result, The project broadens the software application to CO_2 storage area. The main contribution is a new feature where the objective function can be calculated by an external code, e.g., a Python script. This enables an easy way for users to implement fit-for-purpose objective functions. This greatly increases the applicability and ease of dealing with customized reservoir optimization problems in FieldOpt.

To demonstrate the reliability and convenience of the new features, a synthetic CO_2 model is built to work as a base case to be optimized. The *FieldOpt* would jointly optimize the well control and well placement for the CO_2 storage with an injection well and production well. The newly developed optimization framework is tested using the genetic algorithm and the particle swarm optimization method. The result of these tests proves that the new implementation in *FieldOpt* enables a flexible way to study the CO_2 sequestration optimization problem with customized objective function value calculation.

PREFACE

This thesis is written as part of the Master's degree in Petroleum Engineering at the Department of Geoscience and Petroleum at NTNU, the Norwegian University of Science and Technology. It was written during the spring of 2024 under the supervision of Professor Carl Fredrik Berg and co-supervised by Researcher Thiago Lima Silva at SINTEF.

The thesis works on the software - *FieldOpt* - an open-source project initiated by the Petroleum Cybernetic Group at NTNU. The group's team primarily designs and structures the modules of *FieldOpt*, where the contributors are continuously adding new features to the groups repository at GitHub. This tool is also a reservoir management and production optimization project, a research area of the BRU21 - Better Resource Utilization in 21st Century - program owned by the Department of Geoscience and Petroleum NTNU.

The framework of *FieldOpt* has provided a platform to facilitate MSc. and Ph.D. students to conduct research concerning petroleum field development. In this thesis, the software is extended to extend its capabilities towards carbon sequestration, which is a required component in the transition toward the low-carbon energy future. Within this background, this master thesis work develops new features in *FieldOpt*. It has also created a simple CO_2 injection case to serve as a base case for testing out CO_2 injection optimization. The source code and data covered in this report are all available from the commits to the *FieldOpt* GitHub repository:

`https://github.com/PetroleumCyberneticsGroup/FieldOpt`

ACKNOWLEDGMENT

I would like to thank my supervisor, Carl Fredrik Berg, for his guidance and support throughout my master's project. I am also grateful to my co-supervisor, Thiago Lima Silva, for his insights and advice on development work. Without the help and support of both of you, I could not have accomplished this project.

In addition, I would like to express my gratitude to the Petroleum Cybernetics Group, the developers, and the contributors of *FieldOpt*, which enabled me to work on this project and learn a lot from it.

I also want to thank all my classmates, colleagues, and friends who have helped me during these years. Lastly, I appreciate every family member who loves me and makes me who I am.

CONTENTS

Abstract **i**

Preface **ii**

Acknowledgment **iii**

Contents **v**

List of Figures **v**

List of Tables **vi**

Abbreviations and Glossary **viii**

1 Introduction **1**
1.1 CO_2 sequestration problems 1
1.2 Motivation 1
1.3 Research objectives 2
1.3.1 External objective function calculation 2
1.3.2 New component for NPV calculation 3
1.4 Outline of thesis 3

2 Background **5**
2.1 Carbon storage 5
2.1.1 CO_2 trapping mechanisms 5
2.1.2 Modeling for CO_2 storage 6
2.2 Optimization of carbon storage 9
2.2.1 Literature review 9
2.2.2 Well placement 10
2.2.3 Well control 12

3 Methodology **13**
3.1 Software overview 13
3.1.1 Reservoir simulator 13
3.1.2 *FieldOpt* optimization framework 18
3.2 Objective function 22
3.2.1 Net present value 22

3.2.2 Instantiation for CO_2 23
3.2.3 Constraints 23
3.3 Optimization algorithm 23
3.3.1 GA 24
3.3.2 PSO 25

4 Implementations 27
4.1 The *objective* namespace 27
4.2 Update the current NPV class 30
4.2.1 Verifying the updated NPV class 32
4.3 Extend NPV components 33
4.3.1 Employing extended NPV components to CO_2 case 33
4.4 External objective function calculations 36

5 Case Study 39
5.1 Tilted model 39
5.1.1 Well control schedule 41
5.1.2 Well position definition 41
5.2 Optimization configuration 42
5.2.1 NPV components 42
5.2.2 Methods and penalty 43
5.3 Python scripts 43
5.3.1 Python Objective class 43
5.3.2 Python Objective case 44
5.3.3 Path handler 44
5.4 Shell script 44
5.5 Outcome 45

6 Summary and future work 47
6.1 Summary 47
6.2 Future work 47
6.2.1 Improve the case study 48
6.2.2 Diverse objective function 48
6.2.3 Parallel computing 48

References 49

Appendices: 51

A - Github repository 52

B - Nomenclature 53

C - The C++ code 55

LIST OF FIGURES

2.1.1 Overview of a conceptual storage site and different trapping mechanisms. The buoyant CO2 will move towards lower pressure (the surface) (Anthonsen et al. n.d.) 6
2.1.2 Generic reservoir modeling workflow (Ringrose and Bentley 2021) . 7
2.1.3 (a)Rock model (b)property model (c) designed for reservoir simulation for developing planning.(Ringrose and Bentley 2021) 8

3.1.1 Numerical Solution of Equations Schematic (Baxendale n.d.) 16
3.1.2 *FieldOpt Module overview* 19
3.1.3 *FieldOpt* JSON input example 20
3.1.4 *FieldOpt* workflow 21

4.1.1 The previous objective namespace diagram 28
4.1.2 Data flow of NPV component from JSON to *NPV* class 29
4.2.1 time slot example 31
4.3.1 two dimension Model with two well 34
4.3.2 Simulation Model of the 2D model with two well 34
4.3.3 Simulation of Optimal case for the 2D model 35
4.3.4 NPV values versus total interaction numbers plotting 35
4.4.1 New objective namespace diagram 36
4.4.2 New implementation feature in the optimization loop 37

5.1.1 Random field generation data for the model 40
5.1.2 Base case simulation result 40
5.1.3 *WellSpline* coordinates and boundary 41
5.4.1 Shell script command workflow for running facilitation 45
5.5.1 Best case simulation result 46
5.5.2 *FieldOpt* internal and external calculation in three optimal cases . . 46

LIST OF TABLES

3.1.1 Fluid phase type and components of flow simulator 14
3.1.2 The main sections and included data type of the flow input deck . . 17
3.1.3 FieldOpt main feature overview 22

4.3.1 Optimal result for two 2D model 35

5.1.1 well control setting 41
5.1.2 WellSpline setting and constraints 42
5.2.1 NPV component and well cost setting 42
5.5.1 Optimization result from FieldOpt. 45

ABBREVIATIONS AND GLOSSARY

- **NPV** net present value
- **CCS** carbon capture and storage
- **GA** genetic algorithm
- **PSO** particle swarm optimization
- **OOP** object-oriented programming
- **BHP** bottom hole pressure
- **EOR/IOR** enhanced oil recovery/improved oil recovery
- **JSON** javaScript object notation, a text format to store data
- **EOS** equation of the state
- **GUI** graphic user interface

- ***Flow*** a reservoir simulation software
- ***FieldOpt*** a programming framework for field development optimization
- ***Eclipse*** a commercial reservoir simulator
- **well** a hole drilled in the surface for exploration, fluid extraction, or injection
- **shell** a program interprets user command and executes it
- **namespace** a C++ programming concept. The namespace groups the code together and prevents name collision
- **pointer** a C++ programming variable. The pointer stores the memory address of another variable or object.
- **commit** a snapshots or milestones along the timeline of a software project code.

- **parameter** a variable deceleration of the function or command.
- **argument** actual value passing to the function or command.

CHAPTER

ONE

INTRODUCTION

1.1 CO_2 sequestration problems

Carbon capture and storage (CCS), also known as CO_2 sequestration, is an effective method of securely storing CO_2 in subsurface areas, which helps to reduce the concentration of CO_2 in the atmosphere. The geological site is the main determinant for the storage capacity; additionally, the operational schedule for the practical engineering process must also be determined to inject as much CO_2 as possible under the premise of safety. The well drilled from the ground and interconnected with the formation layers is the channel for fluid flow between the surface and reservoir. Therefore, to inject more CO_2, we always have to decide on the well locations and design facilities operation plans with larger potential storage capacity.

1.2 Motivation

Economic analysis is vital to project investment and business. A common analysis objective is the Net Present Value (NPV), which discounts future earnings. The NPV value reflects whether a project is worth investing in, and also how profitable the project is. Forecasting reservoir performance is needed to provide input to NPV calculation.

This thesis deals with modeling, simulation, and optimization of CO_2 sequestration. For a CO_2 sequestration project, this encompasses running many reservoir simulations, which are known to be computationally heavy and therefore time-consuming. This is one hindrance for engineers to search the optimal case manually. Many factors are of concern during the process, from injection sites to migration paths. Besides, sometimes multiple reservoir realizations are used to handle geological uncertainty. These issues entail a large workload and time to make better decisions for CO_2 injection projects. Within this context, *FieldOpt*, as an optimization framework, is a tool that has been designed to increase productivity and effectiveness in the decision-making process of hydrocarbon production projects. There is a similar need for the automatization of modeling, simulation, and optimization of CO_2 sequestration projects. Therefore, there is a need

for extending the current *FieldOpt* abilities to CO_2 sequestration problems. Further, the consideration of case-based problems drives the project to develop an open-accessible code, compatible with various fluid and easy-deployed software interfaces, to define the user-customized optimization evaluation criteria.

1.3 Research objectives

In the pre-stage of a CO_2 storage project, the reservoir model and simulation-based techniques are applied to forecast fluid flow behavior. This can be simulated with reservoir simulation tools, such as the *Flow* simulator used in this project. Simulation results are then analyzed, and the boundary conditions of the model (i.e., well rates, pressures, and constraints) are modified many times to study the possibility of more CO_2 being injected at more favorable injection pressures within pressure boundaries and without CO_2 flowing towards areas where it can leak out of the reservoir. The *FieldOpt* software, which iteratively runs with a reservoir simulation simulator (e.g., *Flow*), automates the process with an optimization methodology to find (a local) optimal solution. This is a foundation for decision-makers to decide on development plans. It is reasonable that the methodologies used for optimizing hydrocarbon production can be reversed to optimize injection, e.g., of CO_2 for sequestration or of H_2 for storage. Though *FieldOpt* was initially designed for the optimization of oil and gas production, with some modification, the software is expected to be able to optimize the injection and storage of CO_2 underground. The simulators required by *FieldOpt* have recently been extended to support CO_2 simulations, such as the inclusion of the `CO2STORE` keyword in *Flow*. It is then necessary to enable CO_2 as a new fluid type to be optimized within the *FieldOpt* software. While the extensions of *FieldOpt* towards CO_2 sequestration developed in this thesis should be generic, the development was centered around the simulation tool *Flow*, created by the Open Porous Media Initiative. The reason for focusing on *Flow* is that this is the only major reservoir simulator with an open-source license, ensuring the transparency sought in research.

The target of this project is to enable *FieldOpt* to be available for CO_2 sequestration. Moreover, the project aims to reduce the effort and time to deploy any form of the objective function for any setting, including CO_2 sequestration, hydrogen storage, or hydrocarbon production, without specific background knowledge of the *FieldOpt* software.

The following subsections outline the two main objectives of this thesis.

1.3.1 External objective function calculation

In mathematical optimization problems, the objective function is the real-valued function whose values will be minimized or maximized. In our geoscience optimization situation, simulation model cases are generated by the optimization loop and will run either sequentially or in parallel. Two types of objective functions are included in *FieldOpt*: The Net Present Value(NPV), a profit function that discounts all the earnings and costs to the present, and the Weighted Sum, where the user can specify weights for different outputs from the reservoir simulation.

Both types of objective functions can handle problems with certain constraints. Both of these objective functions are coded within the *FieldOpt* software and are hard to modify because of the complex syntax of the C++ object-oriented programming language, and any change to the code implies that you need to compile and build the software anew. Instead of providing another form of the objective function to *FieldOpt*, we intend to make the function value accessible by calling a code outside the *FieldOpt* software, then without any restraints on how to modify the function itself.

1.3.2 New component for NPV calculation

By analogy with cash flow in hydrocarbon production scenarios, CO_2 injection gives a new cash flow from income on gas injection. Gas injection was considered a cost in the traditional setup of *FieldOpt*, while optimization of CO_2 injection requires gas injection also to be considered as an income. We will therefore extend the components used for NPV calculations in *FieldOpt* to also contain gas injection as an income source. To verify the correctness and reliability of the modified NPV function and the external objective function calculation, this thesis compares optimization where one calculates the NPV value with the newly added objective type for the CO_2 component with optimization using the NPV calculated externally through a Python script replicating the internal calculations.

1.4 Outline of thesis

The thesis is primarily composed of four sections. The next chapter focuses on the research background of CO_2 sequestration, giving research study cases to describe the carbon storage mechanism, challenge, and objective. Then, we are going to the theme of our thesis - optimization. The general characters and topic of a CO_2 optimization problem are discussed, including the definition, input variables, methodology, constraints, and objective functions.

Chapter 2 will introduce the trapping mechanisms of how the CO_2 is trapped in a brine aquifer. We will briefly explain the general step of modeling CO_2 storage, and also discuss the difference between the hydrocarbon reservoir modeling and CO_2 sequestration. The general modeling design mind is explained in the form of the workflow and investigated in relation to the essential model type. This modeling method instructs what we need to build a model for CO_2 storage and optimization. Two optimization objectives are reviewed to give insight into common optimization issues: well placement and well control. Those two aspects will also be our project's optimization problem target.

The following chapter 3 studies the principle and solution of how the simulator *Flow* and optimization framework *FieldOpt* facilitate the optimization problem. The CO_2-EOR or CO_2 storage model is extended from the black oil model; the simulator implements the Newton iteration and non-linear iteration workflow to advance the simulation time. This is fundamental to introducing the new feature for *FieldOpt* because we post-process the simulation result to evaluate the optimization case. In general, the *FieldOpt* optimization framework executes many

simulation cases and gives out the optimal result. A few topics, including the software input, workflow, and module software architecture, are introduced to acquire the fundamentals for developing new features. Optimization concepts, such as the objective function, constraints, and optimization algorithms, will also be covered in this chapter. Parts of this chapter have overlaps with a previous specialization project leading up to this master thesis.

The implementation is described in chapter 4, were we are conducting the C++ object oriented programming in *FieldOpt*. We use the unified modeling language to display the C++ class, objects, and function. In this chapter, we completed three main tasks. First, update the previous code to remove bugs and errors, and improve the readability of the NPV calculation in the previous source code. Next, add the new NPV component to support CO_2 sequestration optimization in *FiedlOpt*. Then, we design the external objective function value calculation. The NPV component extension is a continuation of work conducted in a previous specialization project, while the other parts are new.

Next we perform a case study in chapter 5 using our newly implemented method to optimize a CO_2 storage case. A CO_2 storage model is created, with a tilted corner-point grid and a Gaussian covariance model to generate the model properties. The optimization of this model is jointly done with the well control and well placement. Additionally, a Python script is created as an external tool to calculate the objective function value (NPV), and a bash script and path handler facilitate the running of this example case by *FieldOpt*. Lastly, we illustrate the result and compare the outcome using different algorithms and objective function types.

The final chapter summarizes the entire thesis work and suggests future improvement.

CHAPTER

TWO

BACKGROUND

Underground geological CO_2 storage is a complicated process and involves many topics. This chapter gives a rough description of the CO_2 sequestration mechanisms and focuses on a unique CO_2 storage than the conventional oil and gas modeling and optimization background.

2.1 Carbon storage

Geological storage of CO_2 is injection accompanied by storage of CO_2 streams in underground geological formations (Anthonsen et al. n.d.). The CO_2 is compressed to the dense form, injected through a well, and stored in the subsurface formation. The conventional oil and gas reservoir modeling is to extract hydrocarbon by means of gas/water injection. However, the CO_2 exists in the supercritical phase in the deep subsurface for storage; more precisely, the liquid-like density and gas-like viscosity phase CO_2 are not intuitive to understand and have many differences with hydrocarbons. Therefore, we have to clarify the basic principle and mechanism in the CO_2 storage processes in the CO_2 - brine systems.

2.1.1 CO_2 trapping mechanisms

The Figure 2.1.1 illustrates the four trapping mechanisms that retain the CO_2 in the reservoir formation. We will explain each physical and chemical mechanism and determinant.

Residual trapping The CO_2 is trapped in the capillary pores of the bring saturated reservoir rock. After injection, the major CO_2 plume moves towards the caprock seal while the minor CO_2 bubble is left behind in the rock pore. The relative permeability hysteresis of the rock matrix and fluid phase highly influence this physical trapping of CO_2, but it is the safest and quickest CO_2 mitigating mechanism (Vishal and Singh 2016).

Structural/stratigraphic trapping The injected CO_2 migrates upwards because the CO_2 is less dense than the brine. The impermeable cap rock prevents the buoyant CO_2 from moving upward. Furthermore, it is because the small grain and poor interconnect pore throats of the low permeable rock usually entail a

strong capillary force and pressure barrier to stop the CO_2 enter or pass through; a large amount of mobile CO_2 is stored in this way in the injection period.

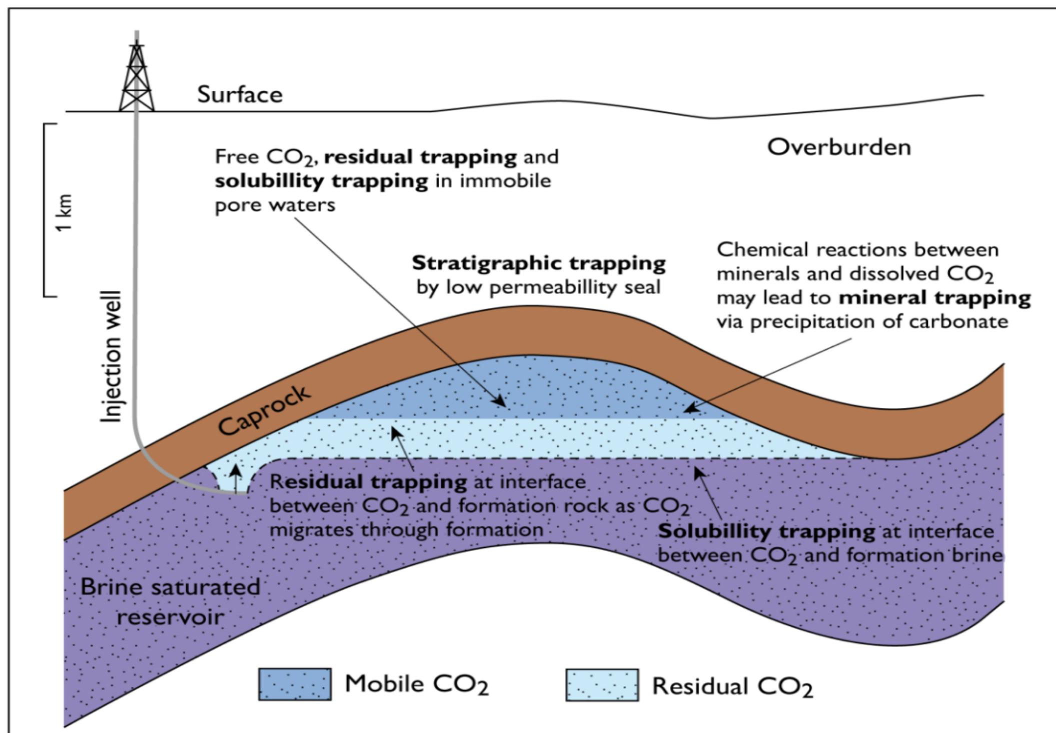

Figure 2.1.1: Overview of a conceptual storage site and different trapping mechanisms. The buoyant CO2 will move towards lower pressure (the surface) (Anthonsen et al. n.d.)

Difussion/solubility trapping The another chemical trapping is gaseous CO_2 dissolution into a fluid phase. The molecular diffusion is the CO_2 molecules move from the CO_2 plume to the brine phase in the aquifer. Temperature affects the diffusion rate, but it is still a slow process. However, the density-drive convention is faster and more efficient. As the CO_2 starts to dissolve into the brine, the CO_2-brine mixing fluid is denser than the pure bring fluid; this causes a further sinking and mixing and dissolution of CO_2. The brine salinity is proportional to CO_2 solubility.

Mineral Trapping/Mineralization The mineral trapping is a long-term storage mechanism where the CO_2 reacts with reservoir fluid or formation to form a solid carbonate minerals. It depends on the rock type, the mineral composition of the rock, and the pore fluid. This geochemical process traps the CO_2 in a solid phase, such as the precipitation of carbonate minerals and CO_2 adsorbed on the clay mineral. This mineralization occurs slowly in nature over geological time.

2.1.2 Modeling for CO_2 storage

In the early days of the petroleum industry, three-dimensional(3D) volumetric modeling and reservoir simulation technology had been developed to determine the hydrocarbon source and predict the flow of the fluids. The modeling of CO_2

storage model can also apply the existing techniques to guide the storage project. But we have to point out that there are fundamental differences between the modeling for CO_2 storage and hydrocarbon reservoir. The CO_2 storage complex needs a much larger rock volume to be considered for the inclusion of storage units, sealing formation and faults, site boundary, monitoring, and so on.

Fit-for-Purpose model design mind (Ringrose and Bentley 2021) is the approach to build a model for the project's purpose. It is not a concrete model but a general guideline to build various types of models, including CO_2 storage as well.

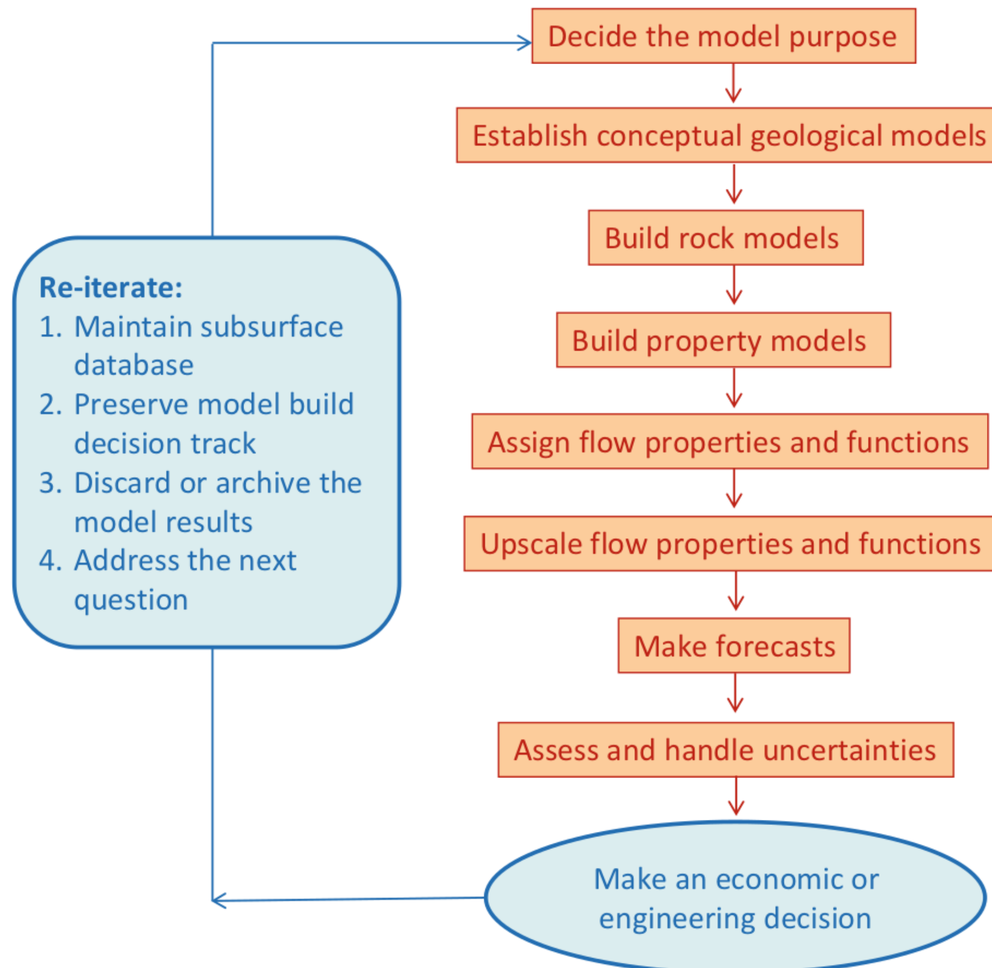

Figure 2.1.2: Generic reservoir modeling workflow (Ringrose and Bentley 2021)

The Figure 2.1.2 gives the work *Flow* of building a reservoir model based on this idea. The workflow includes the common process of building a reservoir model that fits any modeling purpose and the decisive step to make evaluation and engineering decisions.

In our optimization software development work, we must stress that the purpose of the project is to: Create a CO_2 storage model for the CO_2 flow simulation; the model is only used for instantiating the optimization research work. In other words, we follow the procedure to build a simplified and representative CO_2 storage model to implement a new method and carry out economic evaluation work

in the optimization software. We don't touch the model up-scaling, uncertainties, and iteration process in our work as it is enough for our software development work.

Most reservoir models should be able to run with the Front-End simulator. The three types of models in the workflow are displayed in Figure 2.1.3. Those models are built with millions of cells within a 3D model domain. The general description of each model is introduced as follows:

(a) The rock modeling usually captures the contrasting rock identified from sedimentology and displays it in 3D. The rock model is the place to guide the spatial property distribution in 3D. Rock modeling is useful if it helps represent the reservoir properties, but it is unnecessary if the property model is representative enough to simulate. The fault and boundary are the most important geological features to be modeled. In chapter 5, A tilt cubic geological model is generated using a corner-point grid method. This model only sketches the geometry of the model without structure and faults.

(b) The reservoir simulation output is quantitative, and the numerical simulation result is derived from the property model. The *Reservoir model design* (Ringrose and Bentley 2021) proposes that reservoir modeling aims: "To capture the knowledge of the subsurface in a quantitative form in order to evaluate and engineer the reservoir." For CO_2 storage, the CO_2-brine relative permeability, primary and secondary porosity, fluid saturation, brine salinity, and temperature are all common for a CO_2 storage modeling. The porosity and permeability field are randomly assigned by the Gaussian Covariance Model (Müller et al. 2022) to the tilt model in chapter 5.

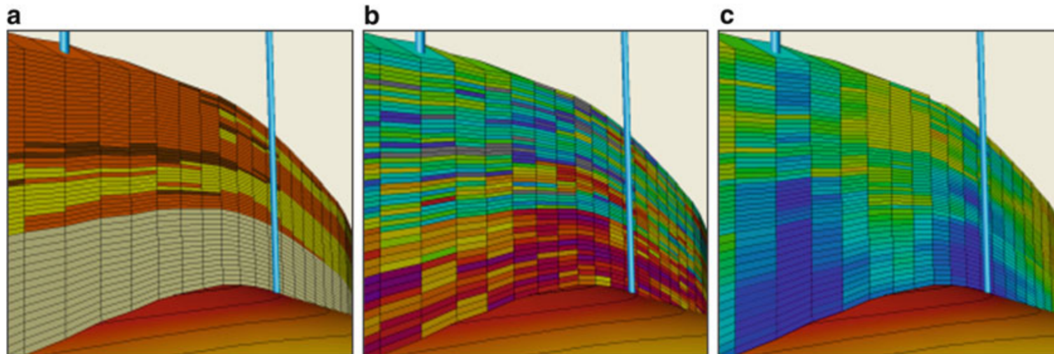

Figure 2.1.3: (a)Rock model (b)property model (c) designed for reservoir simulation for developing planning.(Ringrose and Bentley 2021)

(c) The reservoir simulation for development planning usually requires the model for well planning and fluid recovery. The simulation model could be used to design the well path. The well cost and productivity are assessed as part of reservoir development management to optimize the oil extraction and drill investment. The EOR/IOR process usually includes flooding methods such as water flooding, miscible gas injection, CO_2 flooding, and the injection strategy. This fluid recovery model used detailed sector or near-well models with relatively simple and coarse full-field grids (Ringrose and Bentley 2021). The well, whether for hydrocarbon

production or EOR/IOR fluid injection, is a channel for fluids to flow between the ground and subsurface. Without a doubt, conventional well modeling is also applicable to CO2 storage projects. Therefore, we will not introduce specific modeling methods here; instead, we should focus more on engineering and practical application problems with respect to well. The following optimization section will discuss the decision about the well configuration with the reservoir.

Overall, there are similarities with hydrocarbon reservoir modeling, like finding the flow barriers, pressure compartments, good permeability zones, well modeling, etc. At the same time, other issues for CO_2 storage give a new challenge for reservoir modeling. The short-term and long-term injection for different trapping mechanisms needs a fine and detailed model. Additionally, the understanding of the formation of geochemical and geomechanical reactions due to CO_2 injection and its flow behavior has to be considered in the reservoir modeling design. The CO_2 storage modeling generally requires large volume and sealing integrity of the storage unit with a high-resolution grid and level of detail.

2.2 Optimization of carbon storage

After we have built a reservoir simulation model in our CO2 storage project, the next question is: how can we design, drill, and operate wells to meet the CO2 injection target? How do we find the best way to safely inject as much CO2 into the ground as possible at the lowest possible cost? Studies and research have been performed on the optimization of CO2 sequestration, meanwhile coupling with oil and gas production as a more profitable and practical way for CCS project development.

2.2.1 Literature review

As we discussed before, one of the CO2 geological storage sites is a brine aquifer. There are several trapping mechanisms to store the CO_2 in the subsurface. The study (Cameron and Durlofsky 2012) had optimized the amount of the injected CO_2 in the mechanism of dissolved, residual, and structural trapping; they aim to minimize the mobile fraction of CO_2 at the end of the 1000-year storage operation. The well placement and CO_2 injection well control is the primary optimization problem. Additionally, they also study the well control for CO_2 injection well and production well; the brine production and re-injection (brine cycling) schedule and well control parameter also is optimized in their study using Hooke–Jeeves Direct Search(HJDS) algorithm, and a synthetic aquifer simulation was executed with CO2STORE option in the ECLIPSE reservoir simulator. It is worth mentioning that in order to investigate brine cycling and the optimal time, the object function, including the fraction of mobile CO_2 after 1000 years, at 100 years, and time-averaged mobile CO_2 over 10000-year were tested in the optimization study with and without brine cycling case.

One more specific CO_2 injection study optimized both oil production and CO_2 storage in the Farns- worth Unit (FWU), Ochiltree County, Texas.(Ampomah et al. 2017). The basis of the study was developed with a reservoir model that was

constructed from a geological framework of FWU as a simulation and optimization method. An EOS with laboratory fluid analyses to predict the thermodynamic minimum miscible pressure and history-matching calibration of primary, secondary, and tertiary recovery. The Eclipse compositional simulator was used in the simulation runs. A genetic algorithm approach was employed to determine the optimum development strategy among the control variables: water alternating gas cycle and ratio, production rates, and BHP of injectors and producers. The cumulative CO_2 injection of the defined multi-objective function was computed with proxy models. The work demonstrated a framework to co-optimize CO_2-EOR using a genetic optimization approach to predict the optimal operational parameters while maximizing the multiple objective functions.

We can find the characters of the optimization problem through the study above. The well placement, well control, and case-oriented objective function in those cases have been studied in their optimization work. The following section will expand on the details of the critical factors regarding the well. In those two studies above, the optimization project of CO2 sequestration has the following common characteristics: First, the simulation-based reservoir model works as a basis. Second, the optimization algorithm works as a searching method; third, parameterized reservoir variables with an objective function that works as a target. The *FiledOpt* framework integrates all three features and has extensive support for various problems in this aspect.

2.2.2 Well placement

In the petroleum industry, well placement is the principle of drilling a directional or horizontal wellbore oriented to maximize contact with the most productive parts of the reservoir. The hydraulic and natural fracture is the target location that intersects with the well. To put it another way, well placement determines the optimal location and trajectory for drilling the well with as little cost as possible. CO_2 storage also has a similar concern of carbon injection maximization and drill costs minimization in well placement. The essential factors are as follows:

I. **Geological and geomechanical analysis**

   The geological feature is the primary and decisive aspect for well placement of CO_2 storage. The well planning has to avoid destroying the storage unit integrity and bypass the fault, fold, and natural fracture. The rock stress and elastic deformation caused by CO_2 injection has to be analyzed before the injection. The formation's pressure build-up would potentially create the hydraulic features and is the risk of CO_2 leakage. The good porosity of reservoir formation will give ample space to storage CO_2.

II. **CO_2 injectivity**

   The injectivity is the quantitative assessment of the injection well performance. It is the ability of a well to accept fluid during injection operation and is expressed through the injectivity index(II). The planned CO2 storage well injectivity is also one of the key questions for site assessment. The

general definition for it is:

$$\text{Injectivity Index(II)} = \frac{\text{Injection Rate}(Q)}{\text{Pressure differential}(\triangle P)} \tag{2.1}$$

and interpreted as the injection rate per unit pressure drop between the wellbore and the reservoir.If we assume the vertical well geometry and use the radial Darcy flow equation, the CO_2 well injectivity equation with high gas flow rate (Ringrose, Greenberg, et al. 2017) can be deduced as below without the skin factor:

$$II_{\mathrm{CO_2}} = \frac{q_{\mathrm{CO_2}}}{p_{fbhp} - p_{res}} = \frac{1.406 k_{res} h_i (p/\mu_g Z)}{T[\ln{(r_e - r_w)} - 0.75]} \tag{2.2}$$

The definition for each symbols is in Appendix B.1 Good injectivity indicates efficient and effective CO_2 injection and storage. The high injectivity also provides a flexible injection operation, allowing injection rates to be adjusted without causing excessive pressure build-up and fracturing.

The two factors above give us an intuitive understanding of the effect of the reservoir heterogeneity and characteristic to CO_2 injectivity. Of course, the storage capacity and long-term containment and CO_2 plume migration strategy are also influential factors. In this thesis, we are trying to resolve the issues that impact the well placement: the rock porosity, permeability, and pressure distribution since we use a random field generation of the reservoir properties. In chapter 5, we optimize the well placement using toe and heel well definition methods in the constraint of reservoir boundary in 3D. That is to say, the well placement optimization is confined to the box-shape region. The reason is to ensure that the well is placed enough distance from the production well. The *FieldOpt* automates the process of the optimal well location and completion interval searching in the given area.

III. **Economic factors**

Drilling a well takes up a large portion of field development costs. Drilling costs rapidly increase when drilling wells of a longer length. Achieving and maintaining high injectivity is preferable but also costly in terms of economic considerations. Therefore, the expenditure for well drilling must be balanced against the economic benefits. The *FieldOpt* framework also contains the well cost feature in the well economy C++ class; different well sections (horizontal and vertical) costs are also separated in the calculation. The well economy C++ class is also included in the new feature development in chapter 4. We also enable the well cost calculation through the whole optimization case in chapter 5 with the same configuration. We also give the constraint for well length in our problems.

Well placement is the most complicated problem for optimization; there are plenty of effects and uncertainty on this issue, and it involves various geological, economic, and operational constraints and boundaries. Next, we will go to the well control topic because they are not solely resolved but interrelated for optimization.

2.2.3 Well control

The well control aims to maximize the oil and gas income revenue and minimize production water disposal expenditure by optimizing well operation in a set of times. As we mentioned above, it usually deals jointly with wellplacement problems. From the Equation 2.2, we can tell that the flow injection rate and flowing bottom hole pressure(BHP) are the components to determine the CO_2 injectivity. The well control problem is, in essence, optimizing the injection or production schedule to inject more CO_2. Control parameters are set to BHP and flow rate targets or constraints for a group of wells at specific control times. We have to note that we set the control model for a well, either production or injection, target as the BHP, the constraint applies to the well, and another case is vice versa.

In our case study, we define the maximum and minimum boundary for those control parameters, and it belongs to the optimizer constraint. Namely, the rate and BHP constraint are deployed in different problem fields. When we discuss the control parameter of the well constant or target, it is about the well operational configuration of the reservoir simulation, which is given from the simulator input; however, when we refer to the well control optimization, the optimization constraint is the boundary for those well operational control parameters, and the parameter is set in the *FieldOpt* running input.

Well control has many practical implications in engineering operations; the limitation of the facilities, the prevention of severe washout to the valve of the downhole and surface equipment, and the risk of induced hydraulic fracture by high-pressure build-up are all practical fundamentals to set a reasonable range for the well control variable in the optimization.

In conclusion, the CO_2 storage is a multi-discipline problem, and it requires modeling for the CO_2 storage model, simulating the CO_2 flow process, especially for the long-term storage process. The project design has to take into account many essential and critical factors for optimization and adjust the injection plan according to the economic evaluation.

CHAPTER

THREE

METHODOLOGY

This chapter discusses and describes the methods used in this thesis. It explains what happens inside the black box when the software program runs. The *FieldOpt* framework is introduced, followed by the *Flow* simulator discussion since the *Flow* is the outside program that is executed by the *FieldOpt* framework. The solution methods of simulator *Flow* is also the software development foundation. Then, we will concentrate on optimization concepts and implementation on the *FieldOpt* and explain the objective function and optimization algorithm in detail.

3.1 Software overview

Flow and *FieldOpt* are two software, respectively, responsible for simulating the fluid flow behavior and automating the CO_2 storage optimization process.

The *Flow* is coupled and invoked by the *FieldOpt* framework in an algorithm loop. We will introduce the principle methods of how the software simulates CO_2 storage. The topic starts with the introduction of the CO_2 storage model, the equation's numerical solution, and the necessary data of the input deck.

With the above simulation work, we can further optimize. By introducing the workflow and JSON-format data input of *FieldOpt*, it is more apparent that the thesis research topic is the objective function. The relevant concept is mentioned as it will give an entire optimization scope to realize the importance of the CO_2 storage problem.

3.1.1 Reservoir simulator

The *Flow* simulator is a fully implicit three-phase black oil type simulator created by the Open Porous Media("OPM") initiative. The simulator development is focused on CO_2 sequestration and enhanced oil recovery(EOR). In other words, The simulator supports the simulation of different types of EOR processes, which are used for recovering oil from a petroleum reservoir with methods. Before the general description of CO_2 EOR model and CO_2 storage in the *Flow* simulator, It is necessary to introduce the black-oil model.

I. **The extended black-oil model**

The black-oil model is the reservoir simulation's most widely used flow model. The model is built on three fluid phases and three components. The term usage that refers to the fluid phase and components is mixed and ambiguous in the industry. Therefore, the table lists all fluid phases and components summarized from the *Flow* simulator manual (Baxendale n.d.);

Black-oil model	Fluid phase	Gaseous	Oleic	Aqueous
	(pseudo) component	Gas	Oil	Water
	Description	Hydrocarbon species of vapor form (at surface condition)	Hydrocarbon species of liquid form (at surface condition)	Pore water
CO2-brine model	Fluid phase	Gaseous(Gas)	Oleic(Oil)	
	component	CO2	Brine	
	Description	Supercritical dense of CO2	Salty water	

Table 3.1.1: Fluid phase type and components of flow simulator

In the fluid phase row of Table 3.1.1, the words in the brackets are also used elsewhere to donate the fluid phase. Once the "CO2STORE" keyword is activated, there is an assumption that, internally, the oil phase refers to the brine and the gas phase to CO_2. This is because OPM *Flow* has several extended black-oil models with the fourth component to simulate the EOR process. The CO_2 Standard EOR model uses an extra component by extending the black-oil oil formulation with a fourth component in the simulator by adding a CO_2 component to the gas(Gaseous) phase (Baxendale n.d.), the CO_2-brine PVT and thermal properties are generated internally in the *Flow* simulator to eliminate the user input.

The explanation for this assumption is not complicated to understand. The oil and gas phase is partially miscible, which means that, to some extent, Gas can dissolve into the oil, and oil also can vaporize into the gas phase. The water phase is immiscible for either of them. This gas oil-water is analogous to the CO_2-brine at the subsurface. The solubility of the CO_2 with brine is similar to the hydrocarbon oil-gas miscibility; as a consequence, this assumption allows the CO_2-brine model to be easily implemented into the black-oil numerical equation with minimal effort to change the code.

Now, we can infer from the discussion above that it is unnecessary to add a new fluid phase to extend the CO_2 as a new NPV component. The *FieldOpt* uses an ERT - Ensemble-based Reservoir Tool - wrapper to read the ECLIPSE-type summary files and get the field and well variable from the post-simulation result. Those variables are differentiated by the phase-type of the fluid and extracted by the ECLSummaryReader class function. The more details would be explained in the C++ Class diagram in chapter 4.

We also comment here that our software development work relies solely on the *Flow* simulator, although *FieldOpt* supports other simulator types. For the same reason, subsequently, the following fluid types appearing in the software input data, python objective function class and case scripts, as well as the picture in chapter 5 is set to this default: **Gas represents CO_2 and oil for brine without special remarks**.

II. **Solution of black-oil model equation**

The *Flow* simulator is a fully implicit solver based on automatic differentiation. The general form of the black-oil equation(Rasmussen et al. 2021) is deduced from the conservation of mass, which means the fact that mass would neither arise nor disappear in a volume element. To apply this law for each component α, we can have the mathematical form as follows, and the definition of the terms is listed in Appendix B.2:

$$\frac{\partial}{\partial t}\left(\phi_{\mathrm{ref}} A_\alpha\right) + \nabla \cdot \mathbf{u}_\alpha + q_\alpha = 0 \tag{3.1}$$

There are three main types of terms for this equation: the accumulation term is the mass change over time within the control volume, the mass change for fluid flowing across the control volume boundary is the fluxes term, and the sink/source term represents the adding mass to /removing mass from the control volume. Well is regarded as the primary source term in reservoir simulation.

Then, the Equation 3.1 could spatially and temporally discretize for each cell of the reservoir simulation model, and the residual is defined as the difference between the left-hand side of the equation and zero. The accumulation term $A^{0}_{\alpha,i}$ with superscript 0 are taken at the start of the discrete time step, while other terms without superscript denotes the quantities at the end of the time step. The subscripts i,j,α are the cell index and pseudo component α respectively. This discretization equation is applicable to each component and grid cell.

$$R_{\alpha,i} = \frac{\phi_{\mathrm{ref},i} V_i}{\Delta t}\left(A_{\alpha,i} - A^0_{\alpha,i}\right) + \sum_{j \in C(i)} u_{\alpha,ij} + q_{\alpha,i} = 0 \tag{3.2}$$

What's more, the equation as mentioned earlier can be written in the compact residual form, where y_n is the vector of solutions to the equation after n Newton iterations and $J(y_n)$ is the Jacobian matrix.

$$J(y_n)(y_{n+1} - y_n) = -R(y_n) \tag{3.3}$$

The solution to this equation is separated into outer Newton iterations and inner linear iterations; the Figure 3.1.1 below illustrates the iteration process to solve the equation.

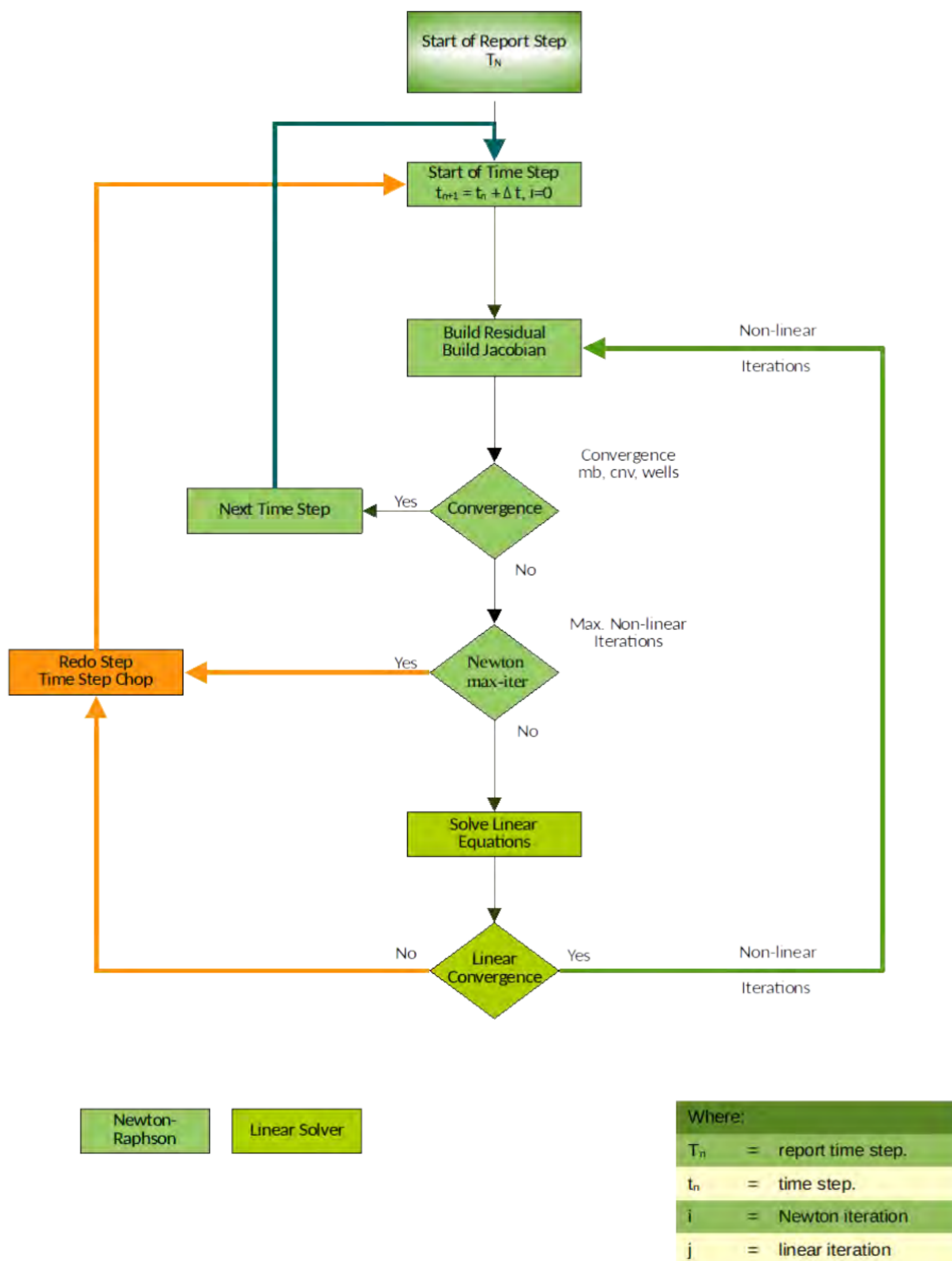

Figure 3.1.1: Numerical Solution of Equations Schematic (Baxendale n.d.)

We are not going further for the mathematical method discussion of the Newton iteration and automatic differentiation to solve this nonlinear equation. The detailed expression for every term of the three phases and the well model is well-explained in the OPM Flow manual (Baxendale n.d.).

So far, we have a preliminary understanding of *Flow* and have enough knowledge of how the simulator works. This flow diagram clarifies that the simulator is advancing simulation based on the time step, and convergence is not guaranteed at a given time step. The time step chops technique is applied when the maximum iteration time is reached and the solver fails to converge. Let's say we have 365 days as the time step, and the simulator is not achiev-

ing the convergence after the maximum Newton iteration. The *Flow* will cut the time step, for example, into 30 days and restart the iteration. If the second time step, 30 days, fails to achieve in the next iteration, the second time step would probably chop into 15 days. This process would continue when the convergence happens or the time step is too small to chop. It is noteworthy that the solver does not always have an integer time step chops into the simulation.

The reason that we discuss this solution is because it is critical and a prerequisite to grasping the iteration method with respect to this solver; the bug fixing and improvement in chapter 4 rely on figuring out the way of advancing simulation time and differentiating the confusing concepts about time and time step for *FieldOpt* NPV class code programming.

III. **Sections of *Flow* input**

The last part of the *Flow* simulator is the input deck for the software. The model file with the extension -.DATA- is divided into several sections to organize the input data, and *FieldOpt* optimization framework would write in the new setting and update the number with a predefined program. The overview of the key section and data type input is displayed on Table 3.1.2:

Sections	Data type
RUNSPEC	- phase present - dimension of the grid - simulation start date - unit system
GRID	- geometry of the grid - petrophysical properties
PROPS	- flow parameters - rock properties - PVT tables
SOLUTION	- initial pressure at specified depth - fluid initial saturation, solubility
SCHEDULE	- well completion setting - well control settings - well state(open/shut) - **simulation advancing time or date**

Table 3.1.2: The main sections and included data type of the flow input deck

As we can see from the table, the key data for modeling a reservoir simulation are divided into different sections. In chapter 2, we have introduced above that we build a synthetic model with the reservoir properties data, and OPM *Flow* has an internal function to create the PVT data table for

CO_2 storage. The schedule section content is more noteworthy for our optimization problem. This section specifies the field development and operation strategy, advances the model over time, and changes the well states. The operation action events are specified at the time when they happen.

In fact, the schedule section is the place where *FiledOpt* optimizes the parameter; the optimization framework has the optimizer that will update the BHP or flow rate constraints and target arguments at a set of control time; the simulator *Flow* is parallelly or serially run with many model cases with different schedule data and remains the rest of section same.

3.1.2 *FieldOpt* optimization framework

Reservoir development decisions are made in an attempt to increase hydrocarbon resources from the subsurface. Similarly, the CO_2 storage is also aimed at utilizing underground storage space and obtaining revenue from carbon reduction under the premise of safety.*FieldOpt* is initially designed to provide an effective tool for decision-making problem about petroleum field development involving the conventional fluid oil and gas, while CO2 injection plays a new and crucial role on the reservoir engineering nowadays for reducing the concentration of CO2 in the air though carbon capture and storage(CCS) technology. The current workflow implemented in FieldOpt allows for improved decision-making and productivity in different applications without substantial changes to the core code.

i. Module Overview

FieldOpt is a modular architecture using the C++ OOP framework, which makes it feasible for the adaptation and extension of the C++ code to fulfill the new feature for CO_2 sequestration; we firstly introduce the four modules and the associated functions in our work. The Figure 3.1.2 diagram shows the relation between the modules.

Model: The *model* module contains the variables that represent the field development plan as well as the properties needed for simulation. The *model* class is also the container of the well object. We have the economy c++ class that is responsible for well cost calculation in the *model* class.

Optimization: All the features related to optimization are grouped into the *optimization* module, i,e, algorithm implementation, constraints, and objective function. We will dive into the explanation of the *objective* C++ namespace and also the objective and its child class. The function *value()* in those classes will execute the objective function calculation.

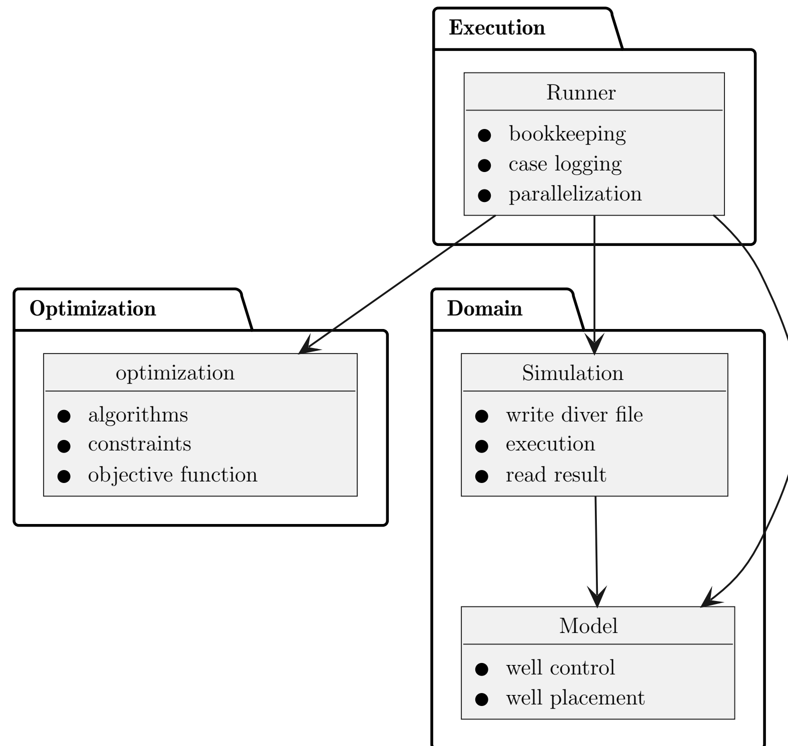

Figure 3.1.2: *FieldOpt Module overview*

Simulation: The *simulation* module is an interface to execute the simulation and read the result. *FieldOpt* launches the simulation work by shell, which provides the creation of a customized shell script. This approach increases the flexibility to pre- or post-process the simulation input and output. We take advantage of this feature by adding a new line of bash script to post-process the simulation result and calculate the objective function value.

Runner: The *Runner* module is to drive and control the *FieldOpt* running in both serial or parallel running. It also deals with the works for logging the optimization case, bookkeeping, and reservoir realizations. Due to the limitation of computing resources, all our case optimization methods are serial in our work.

ii. Software input

The primary input to FieldOpt is a JavaScript Object Notation file(JSON) that specifies the settings for control variable searching space, algorithm parameter, simulation configuration, properties component, etc. Figure 3.1.3 is the part of the data input from the case study showed as a UML JSON diagram. We have to point out that it is error-prone to have a non-compatible data type, symbol-like commas, and incomplete elements for the JSON driver files as a reason for failing to run FieldOpt. A graphic user interface tool (Rykkelid 2016) could be developed to generate the *FieldOpt* JSON input file.

FieldOpt JSON input				
Global	BookkeeperTolerance	1e-8		
	Name	CO2OPT		
model	ControlTimes	0		
		1460		
		2920		
	Reservoir	Type	flow	
	Wells	Controls	BHP	210
			IsVaribale	true
			Type	Gas
			Mode	Rate
			Rate	4550
			TimeStep	2920
		DefinitionType	WellSpline	
		Group	G1	
		Name	INJ1	
		PreferredPhase	Gas	
		SplinePointArray	IsVariable	false
			x	33.0
			y	33.0
			z	1722.5
		Type	Injector	
		WellboreRadius	0.1905	
Optimizer	Constraints	Max	180	
		Min	80	
		Type	BHP	
		Wells	PROD1	
	Mode	Maximize		
	Objective	NPVComponents	COMMENT	
			Coefficient	15
			DiscountFactor	0.08
			Interval	Yearly
			Property	CumulativeGasInjection
			TimeStep	-1
			UseDiscountFactor	true
		Type	NPV	
		UseWellCost	true	
		WellCost	7500	
		WellCostXY	10000	
		WellCostZ	5000	
	Parameters	MaxEvaluations	1000	
		ExpansionFactor	1.0	
		ContractionFactor	0.5	
		InitialStepLength	50	
		MinimumStepLength	5	
	Type	Compass		
Simulator	ExecutionScript	bash_flw-bin		
	FluidModel	DeadOil		
	ScheduleFile	include/2D2WMODEL_SCH.INC		
	Type	Flow		

Figure 3.1.3: *FieldOpt* JSON input example

From the diagram, we can see four main parts of the data structure in the first layer: global, model, optimizer, and simulator. Actually, the *setting* class and its child class parse the optimization configuration arguments from the JSON drive file, initialized as a C++ object and variable, and then processed further in another place, chapter 4 will give a data flow diagram from the JSON input to the objective class to perform the calculation.

iii. *FieldOpt* workflow

The general *FieldOpt* workflow is shown by the Figure 3.1.4; this is a sim-

plified flowchart that omits parallel computing, handling of a constraint violation, and many other features.

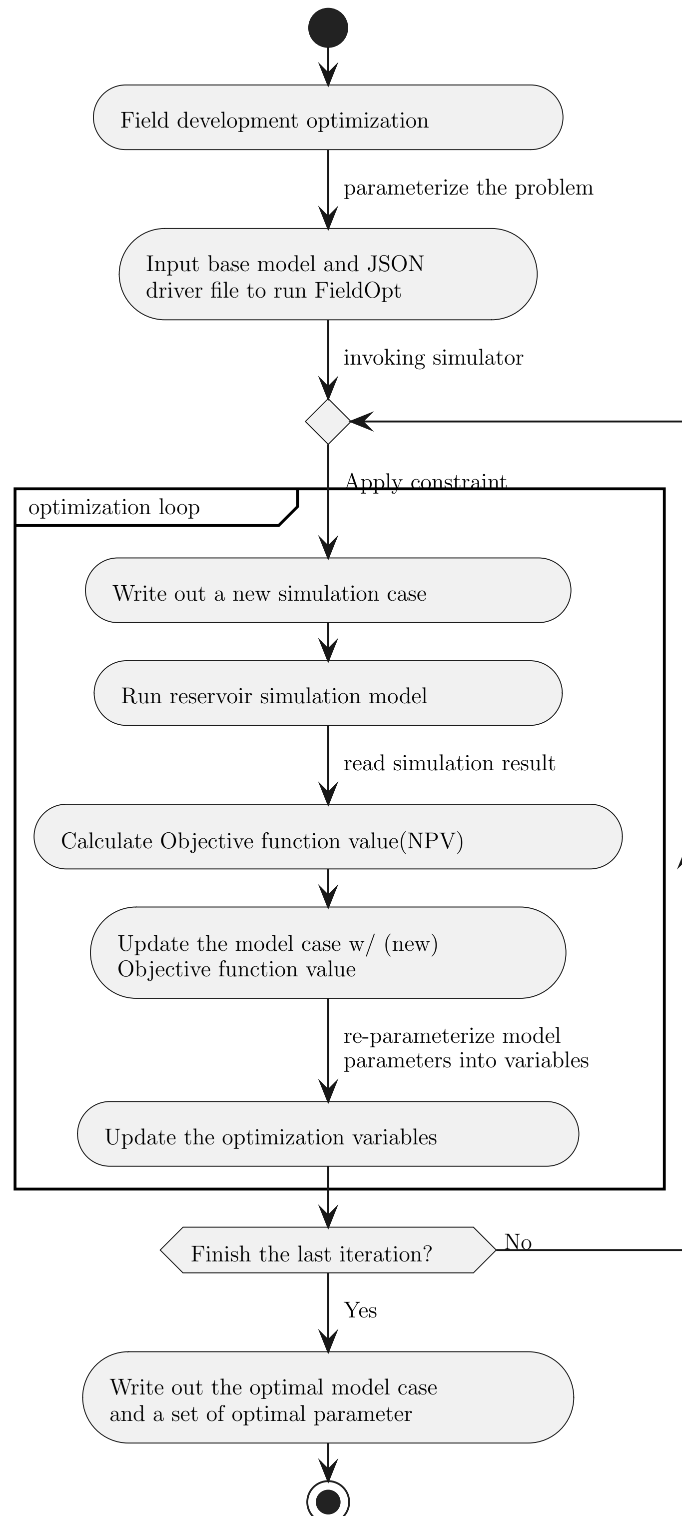

Figure 3.1.4: *FieldOpt* workflow

This optimization process is also suitable for CO_2 optimization. We could tell that the framework - the optimization loop - is the common process for field development optimization. As long as the simulator supports the CO_2 flow,

extending the *FieldOpt* framework is feasible and worthwhile, aiming to solve the problem related to the geological storage of CO2 through this powerful tool. The following section will explain the procedures of the optimization loop.

iv. Supported features

The existing features (Baumann, Dale, and Bellout 2020) that have been implemented in *FieldOpt* are re-arranged from the Table 3.1.3. There are more features that we do not cover in this thesis. The New objection function type - External result - is the third type that is the interface for the objective function calculation outside the *FieldOpt*.

Name	Types	Main function
simulator	ECLIPSE	Control, well placement, completion
	FLOW	Control, well placement
	AD-GPRS	Control, well placement
	INTERSECT	Control, completion
algorithm	APPS	Asynchronous parallel pattern search
	Compass Search	Parallel computing
	Genetic algorithm	Parallel computing
	Particle swarm optimization	Parallel computing
objective function	Weighted sum	Minimizing or maximizing by optimizer
	Net present value(NPV)	
	External result	

Table 3.1.3: FieldOpt main feature overview

3.2 Objective function

The objective function, also known as the cost function or fitness function, is the function that needs to be optimized (minimized or maximized). It quantifies a solution's quality or performance. Consider a function $f : \mathbb{R}^n \to \mathbb{R}$. the function f takes an n-dimensional vector $\overrightarrow{x} = (x_1, x_2, \ldots, x_n)$ as input. Let's say we have parameterized our CO_2 optimization problems as the vector $\overrightarrow{x}$, the CO_2 optimization research study in the literature review of chapter 2 imply that objective function is a flexible and case-oriented function to get the extremum $f(\overrightarrow{\mathbf{x}})$ or a vector of extremum $\overrightarrow{\mathbf{x}}$ that is the representation of the optimal decision.

3.2.1 Net present value

One of the common objective function types is NPV; the NPV calculation usually combines two summations with respect to time and components. The NPV component is the object that brings either positive or negative cash flow. We have both yearly and monthly time intervals for discounting the cash flow; there could be different discount rates and prices regarding the distinct time step for each NPV component. For this reason, there is no standard and unified definition for the NPV calculation. It could be said that we have to discretize a continuous

cash flow into many parts according to the variation of discount rate or price and integrate all in a given time step, then we sum the net present value within the time range. In this situation, however, the calculation of the final result based on the time is more straightforward, which firstly sums all the discounted cash flow in the time range.

In this project, we assume that we have a consistent discount rate and time interval among all the NPV components, and the price per volume unit is constant at the time range for field operation. The NPV formula could be given as follows:

$$NPV(\overrightarrow{x}) = \sum_{i=1}^{N}(\sum_{t=1}^{T} \frac{V_{i,t} \cdot P_i}{(1+r_i)^{t-1}}) \tag{3.4}$$

the definition of the symbols can be found on Appendix B.3

3.2.2 Instantiation for CO_2

We initially display a more general form of a single objective function for possible components in CO_2 optimization problem as follows:

$$\begin{aligned}\text{Objective function} =& \text{NPV} + \text{well cost} + \text{one-time constant} + \text{constraint violation}\\ &+ \text{leakage penalty} + \text{others}\end{aligned} \tag{3.5}$$

We declare that all the NPV occurrence of this project is confined to the cash flow about the earning and cost of flow fluid. The well cost calculation has been coded in the NPV calculation of FieldOpt. The one-time constant such as initial investment, regular maintenance and overhaul for equipment in the future that can also discounted to present time are fixed cost at a give time for CO_2 injection project. The most notable and essential term for CO2 is the penalty of leakage, it can diverse the objective function and geological concern for CO2 sequestration project in different situation. It can put the bound constraint for pressure with regard to impermeable rock lay or structural features, confined CO2 plume migration space in CO2-EOR projects to avoid inter-well CO2 flow, etc. The last term could combined with potential factors about CO2 trapping in the reservoir like salinity of water, underground temperature, and so forth.

3.2.3 Constraints

The variables $\overrightarrow{x}$ are also called decision variables that are subjected to the constraints. As we mentioned in chapter 2, the well control parameter and well placement often apply the optimization constraints. The region where all the variable points that satisfy the constraint is called the feasible region; the *FieldOpt* implements constraint to the candidate solution before they are added to the evaluation queue. Once the constraints violation occurs, the abstract *Constraint* class applies a penalty for each violated constraint and repairs the solution to the feasible space, and the variables will be updated within a pair of bound vectors by the algorithms.

3.3 Optimization algorithm

As we see from the Table 3.1.3, The *FieldOpt* has implemented several algorithms in the optimization framework. There are two evolutionary algorithms in the table; we are

going to choose those two algorithms for our project. one of the reasons is that it is suggested if the simulation time is prohibitive as the unavailable of parallel computing, the GA and PSO will find a good solution in relatively few iterations (Baumann, Dale, and Bellout 2020), In the chapter 5, we also explain further that the choice is made on our optimization problem target and constraints. As we just applied those two methods to our study, we only brief the general idea behind the method and give a pseudo algorithm to explain how it works in the *FieldOpt*. More pre-defined parameters in the software will also be mentioned here.

3.3.1 GA

A genetic algorithm is a search method that mimics natural evolution. The algorithm starts from the random generation of individuals in a population, and a set of individuals is selected according to the fitness of their solution. Then, those fittest individuals will evolve to produce the offspring of the next generation through operations such as mutation, selection, and crossover. This process iteratively updates the individual and converges toward the optimal solution. It will terminate when a satisfactory solution has been found or the maximum iteration numbers are reached, and the fitness function is often the objective function of evaluating the individual.

We first give the important concepts of this algorithm as follows:

- *population*: A set of potential solutions to the problem. The initial population can be randomly distributed.
- *chromosome*: a single solution represented by a set of values.
- *gene*: gene is part of the chromosome, usually on behalf of the attributes or features of the values set.
- *selection*: a process of choosing the individual to create the spring. The solution is assessed by the objective function.
- *mutation*: The operation of randomly changing the individual genes maintains a wider search than the local optimum space.
- *crossover*: also called recombination. The crossover operation combines the genes of two individuals and produces a new offspring. This process ensures that the offspring inherits the best traits from each parent individual.

Then, based on the idea, we can have the pseudo-algorithm for the genetic algorithm;

Algorithm 1 Genetic Algorithm

```
 1: Initialize population with random individual distribution
 2: Evaluate the fitness of each individual in the population
 3: while termination condition not met do
 4:     Select parents individuals from the population based on fitness
 5:     Crossover operation to produce offspring
 6:     Mutation operation to the offspring
 7:     Evaluate fitness of offspring
 8:     Select individuals for the next generation
 9: end while
10: Return the best solution found
```

In *FieldOpt*, we define the probability parameter for the crossover and mutation operation occurrence. The upper and lower bounds are used for the random generation of the population and mutation. Other terms are also implemented; the decay rate refers to the strategy that the mutation or crossover decreases over time; this makes the search initially start from the wide space and then focus on exploring the best solution found. The mutation strength determines the extent of the operation applied to the individual in the population. The stagnation limit refers to the maximum generation for which the best solution in the population does not observe significant improvement, and it often serves as a termination criterion. The last parameter is the discard parameter, which specifies the fraction of the individual that is going to be discarded during the selection.

3.3.2 PSO

The particle swarm optimization(PSO) is implemented as another population-based optimization in *FieldOpt*(Panahli 2017). This algorithm is inspired by the social behavior of birds flocking or fish schooling. The algorithm tries to find the best optimization solution based on the movement of a group of particles. Each particle has a velocity and position, and velocity is a vector that represents the direction and speed at which particles move; the movement is based on the velocity that is updated to the best position of the entire population and its own best-known position. The particle position is iteratively improving in the process until the convergence or the termination criteria are met.

Likewise, we also give the concepts of the PSO method:

- *particles*: The potential solution to the optimization problem. each particle has its own position and velocity.
- *global best*: the best solution found by any particle in the entire swarm.
- *local best*: also called personal best. It is the best solution that has been found by itself.

Algorithm 2 Particle Swarm Optimization (PSO)

```
1: Initialize particle velocity, personal and global positions in a population
2: while termination criterion is not met do
3:     for each particle do
4:         Evaluate the current position fitness
5:         if current position is better than personal best then
6:             Update personal best position
7:         end if
8:         if current position is better than global best then
9:             Update global best position
10:        end if
11:        Update particle current velocity with the previous velocity and position
           with both personal and global best.
12:        Update the particle position with newly velocity
13:    end for
14: end while
15: return global best position
```

It is the same in that there are extra parameters for the PSO to control the particle behavior; the learning factor C_1 and C_2 are the coefficient that influences how much the particle learns from its own history and the entire population. If C_1 is set too high compared to the C_2, the particle movement will be strongly dependent on their personal position, which does the search in a nearby space of the particles. while the high C_2 would result in a premature convergence since all the particles move toward the global best. The inertia weight controls the exploration and exploitation of the algorithm by adjusting the influence of the previous velocities in the current velocity update; the optimization will diverge if the particle velocity is too high to move back to the optimal points. The *FieldOpt* implemented a linear decreasing strategy that inertia is decreased over iteration, which is the strategy to maintain the balance between exploration and exploitation dynamically; the maximum and minimum bound for the inertia must be given if this option is enabled.

CHAPTER

FOUR

IMPLEMENTATIONS

In this chapter, we dive into the C++ OOP of *FieldOpt*; we use the UML language to illustrate the class relationship and data flow. Then, we introduce the bug fixing and development work. The light blue color in the diagram represents the previous features implemented in *FieldOpt*, and the light green is for the newly developed features from this thesis. The footprint of contribution and modification have been merged to *FieldOpt* Github repositories by commit; we will directly refer to the commit name that shows and compares the change for the code script in the sections.

4.1 The *objective* namespace

The class diagram Figure 4.1.1 is the abstraction and structure of the header file within an *objective* namespace. We will diagrammatically explain the picture from the top to the bottom within *Objective* namespace. the source code file is *FieldOpt* Github the directory: *FieldOpt/Optimization/objective*

I. ***Objective class***

The ***Objective*** abstract class brief a virtual function ***value()*** calculates the objective function value that we are going to optimize. It is meant to be overridden by either of the derived class functions with the same name as the class name. Constructors function ***objective()*** is empty because the child class would initialize its variables by its constructor.

II. ***Weightedsum and NPV class***

The class ***WeightedSum*** and ***NPV*** are inherited from the *Objective* class. There are common elements for the two derived classes in the private section. We take ***NPV*** class as an example to illustrate the identical objects and variables shared by both classes. The ***NPV*** class declares four pointers with the symbol *. In the private section, two nested classes with the same name, Component, are declared by ***class*** and ***struct*** specifier in the C++ source code file, and we give the class name in front of them in the diagram to distinguish them. Within the ***NPV-component*** class, it declares a set of member variables of NPV component and member function processing the variable for NPV calculation within the ***NPV*** class. The four pointers mainly work for:

components_: The ***components_*** is the pointer to the QList object. The QList object contains several pointers, and those pointers are used to store the different NPV components in the ***NPV*** class.

result_: The ***result_*** is the pointer that provides access to the simulation result.

well_economy_: this pointer is used for the calculation execution of the well cost.

setting_: The *setting_* pointer is to pass the NPV component arguments from *NPVcomponent* in the ***setting*** C++ class object to the ***NPV*** class.

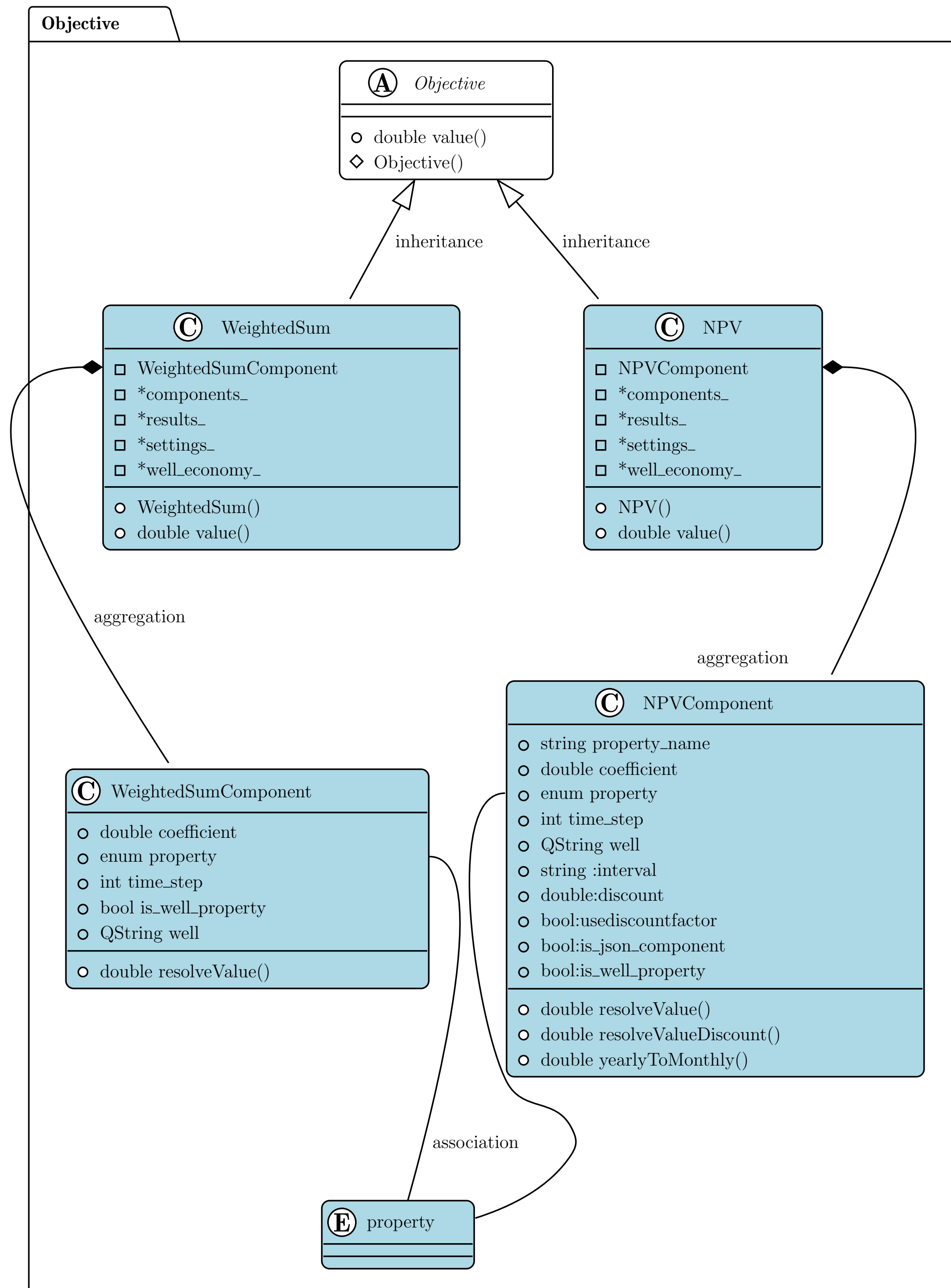

Figure 4.1.1: The previous objective namespace diagram

III. ***Constructor*** Constructors functions are the functions that have the same name

as the class name; ***WeightedSum(), NPV()*** are declared with the public access of both derived classes, which is called automatically when the class object is instantiated. The function initializes the variable value passing outside the class itself, which takes the declared pointers above except pointer ***components_***. The shared enumerated type object **Property** is the data structure used to specify the fluid volume flow type and data retrieval in the *result* namespace. In the *result* class, One of the associated functions ***GetPropertyKeyFromString()*** in ***result*** class that takes the argument data of enumerated type object **Property** is the place where we have to update for extension of NPV component to CO_2.

IV. **Data flow to *NPV* class**

To understand how the NPV component value passes through the JSON file into the objective, Figure 4.1.2 is the diagram that explains the footprint of the input data being stored and parsed to the target destination. This diagram also contains the source code in the *FieldOpt/Settings* directory of the GitHub repository.

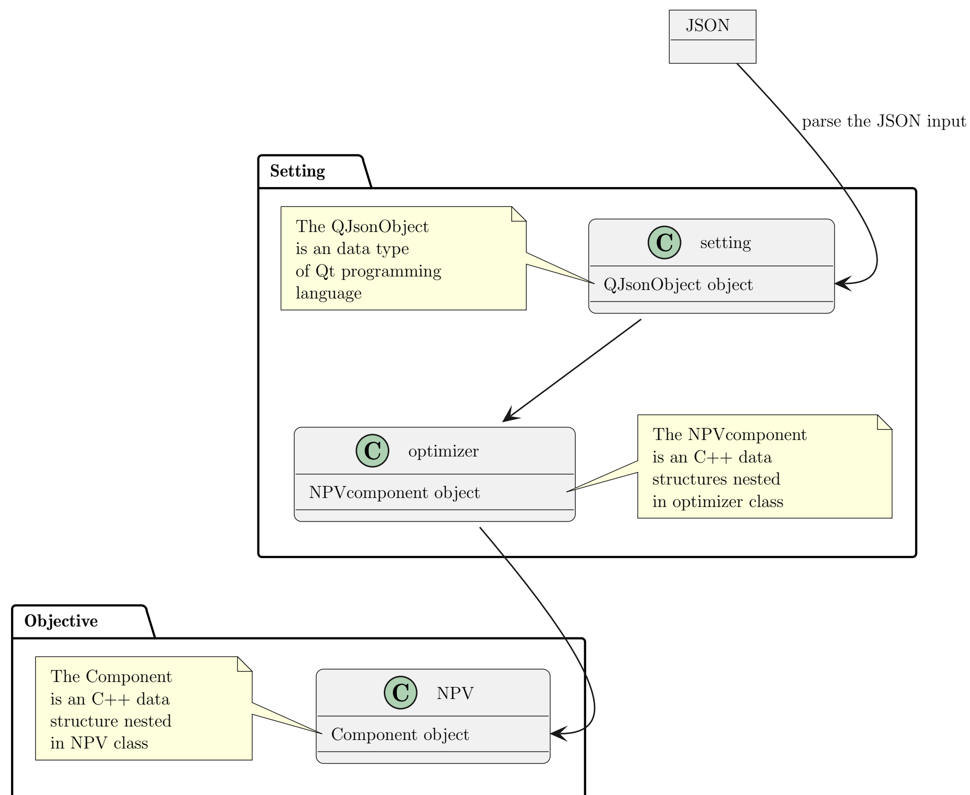

Figure 4.1.2: Data flow of NPV component from JSON to *NPV* class

We can see three data structure types in the whole process. As we explained, the JSON data type is given at the *FieldOpt* input deck. Then, in the ***setting*** namespace, we create a QJsonObject to parse the data input from JSON. The incoming data value is parsed to C++ objects of C++ struct ***NPVcomponent*** type. The child class *Optimizer* has declared the QList type object ***NPV_sum*** that stores those C++ objects of NPV components.

Then, as we mentioned above, we use the ***setting_*** pointer to refer to the QList object ***NPV_sum*** and assign the value to the *NPV* struct object ***Component***.

> The NPV component data is dynamically stored in the object pointed by the pointer ***components_***.

In conclusion, the ***objective*** namespace class and object only perform two tasks: first, use a pointer to parse the NPV component values and dynamically store them within the class object, second, execute the objective function calculation in either of the child class and overwrite the value to the parent class. The reason we clarify this is that these are traits of C++ OOP; the implementation of the new objective function remains these traits, but we do not have the objective function calculation inside the ***objecticve*** namespace, and the only argument that we need to parse from the JSON is the external result file path. Since we only have an absolute file path, we directly assign the file path argument from object to object without using the pointer.

4.2 Update the current NPV class

This section fixes the error in the *NPV.cpp*. The error is that the previous code fails to handle the time vector and mismatches the loop index. Before changing the code, we must first clarify the method and the confusing time-related variables. The previous code has poor readability for the NPV calculation and uses the same index variable in the loop.

Based on the solution of the black-oil model in chapter 2, the terms used in the simulation and NPV calculations are the following:

- *time step*: The time step is the current Euler step that solves the black-oil equation in iteration.
- *report time*: The report time is the time point at which the simulator tries to converge.
- *report time step*: The step length of the report time. It is specified in the schedule section of the *Flow* simulator. The report time length is usually set to 365 days.
- *NPV time*: The time point when we discount all the cast flows into the present.
- *NPV time step*: This step is also the NPV time interval; we have yearly and monthly intervals for NPV discount.

Now, let us consider an NPV calculation case in ten years as an example. Both the simulation report time step and NPV time interval are yearly. The Figure 4.2.1 displays 10 time slots with 11 time points in each axial. In the first two years, the simulator can converge on each planned report time t_i. In that case, we have one-to-one matching between the report time and NPV time T_i shown by the grey arrows. Then, we have the time step chop, and there is a slight deviation between the actual converged report time - the light blue point and the pre-defined time - the red points from t_3 to t_7.

let us set t_i a real value that is equal to the total number of days that the simulator has advanced from the start time, the report time from t_3 to t_7 is not the integer multiple of 365. As a result, the NPV calculation has an issue with solving the mismatching of the two types of time vector data. In the previous code script, the method was to check if the remainder of the dividing t_i by 365 is equal to 0. This is problematic because all the cash flow will not be discounted to the present if the criteria are not met. At the same time, the discount rate is calculated incorrectly and is not applied to the corresponding NPV time interval.

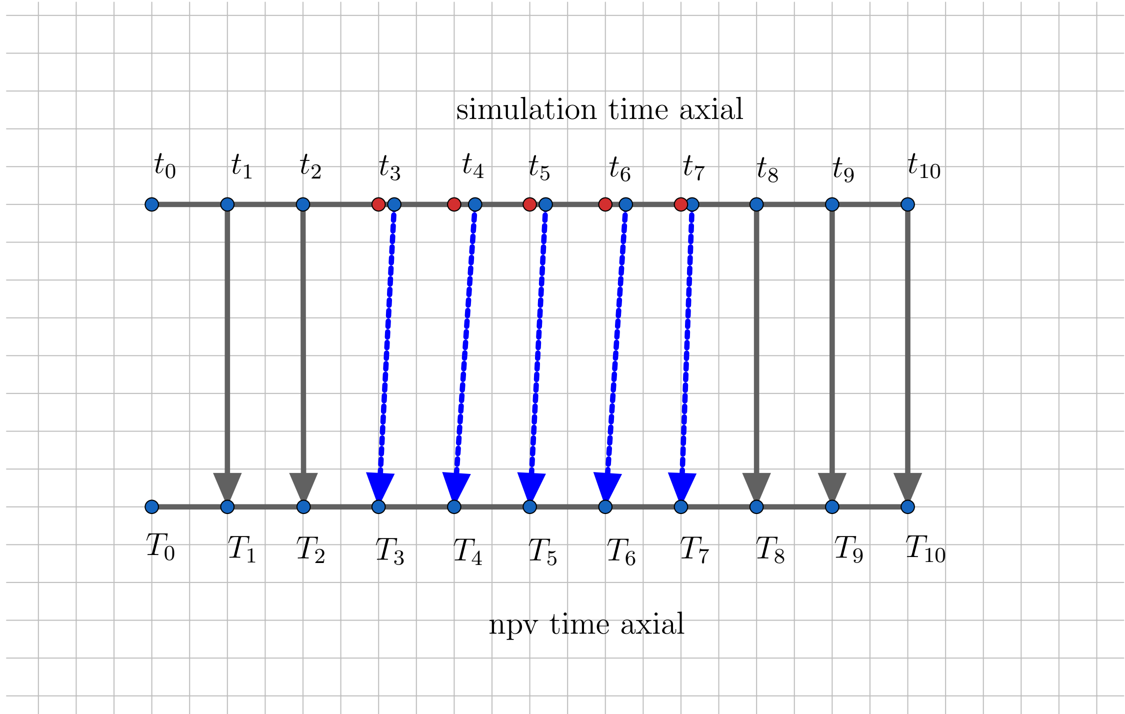

Figure 4.2.1: time slot example

To fix this error, we also use the modulo operation, but we directly calculate the report time index of the i that is defined as:

$$i = \frac{(t_i - t_i \bmod 365)}{365} \tag{4.1}$$

In this way, we project the values from the t_3 to t_7 to T_3 to T_7, showed by the dash blue arrows. We can see that each t_i matches the T_i in two time axials by fixing the mistakes. We also use this index to create the discount list for all the NPV time intervals.

The code bug fixing and improvement can be found at the commit:" Fix the bugs and improve the code for NPV class" in the Github repository. the NPV calculation could also based on the monthly interval, and the method is also applicable. It has to be noted that this is only the approximation of the NPV calculation. we can see that there is still a length difference between the adjacent NPV time and the cash flow from the corresponding adjacent report time. This is the disadvantage of the method, and it is preferred to have a small time step and NPV time step to calculate the NPV values.

Another solution might be that we calculate the discount rate based on the report times. In other words, the discount rate is the function of the adjacent report time step, and we can express the discount rate at NPV time as the form:

$$r_i = f(t_i, t_{i-1}) \tag{4.2}$$

Overall, the NPV calculation is subject to the simulation report times. The last gives the pseudo-algorithm of the improved NPV calculation with yearly time intervals; this algorithm drops the well cost calculation and another error message.

Algorithm 3 NPV calculation

1: Get the report time vector
2: Declare the NPV time list
3: Declare the NPV report time list
4: Declare the discount factor list
5: **for** each NPV component **do**
6:    **if** the component discounted interval is yearly **then**
7:       Declare the discount rate variable
8:       **for** each report time in the report time vector **do**
9:          calculates the report time index i by Equation 4.1
10:          **if** the report time is not in NPV report time list **then**
11:             append the current report time to the NPV report time list
12:             calculates the discount factor based on i
13:             append the discount factor to its list
14:          **end if**
15:       **end for**
16:    **end if**
17:    **if** the component will be discounted **then**
18:       **for** each report time in the NPV report time list, each discount factor in its list **do**
19:          calculates the production difference in the NPV interval
20:          discount all the cash flow into the present and sum up
21:       **end for**
22:    **end if**
23: **end for**
24: **return** the NPV value

It is noteworthy that the general method in the algorithm we observe here is similar to the objective function calculation in the chapter 5. The only difference is that the *resdata* Python package has the function of taking the data keyword and interval, which can directly return the fluid flow volume data. In the next section, we will use this as an external reference to verify the correctness of the modified C++ NPV calculation.

4.2.1 Verifying the updated NPV class

In order to verify this NPV calculation method, we used the base model case - 2 well in two dimensions to calculate the base case objection function value with the GA methods. The details of the model and well control optimization will explained in the next section. Here, we only focus on the volume difference and the corresponding discount rate in that time interval since only two data sets make the difference in the result. The other objective function type - the external result - is independent of the C++ NPV class. Actually, the base case objective function calculation is not related to the algorithm and the model type, and the only requirement to verify is that we have the same NPV component price for different objective function types. The debugging message from the printout of the *FieldOpt* is reprocessed in the ASCII (*.txt) file at the directory of the mater appendix repository: CO2WaterOpt
From the two debug message files, we can see that the base case from the NPV C++ is -37124269613.103569, and the value is -36812594726.509727 for the external methods. The two values are closed, and if we take the Python external methods as the reference,

the difference only takes up 8.5‰ of the reference number. It is accepted, and the reason for this is that the volumetric data is retrieved with different third-party packages.

4.3 Extend NPV components

As we said in chapter 3, the *Flow* simulation result is read by the ERT tool that retrieves the data through the fluid phase. Since the *Flow* simulation result file is an Eclipse-type summary file, we directly navigate to investigate the code block of the ***eclresult.cpp*** and ***result.h*** in appendix C.

In the ***eclresult.h*** code block, we can see that both the well and field type volume flow data are retrieved by the overload function ***GetValueVector()***; the shared common parameter ***prop*** is the enumerated type object ***Property***. the ***prop*** matches one of the cases, and the ***summary_reader*** will read the corresponding fluid volume data.

We can also see that in the ***result.h*** files, the ***Property*** data structure has all the elements listed in the ***GetValueVector()***. Now, the only problem is that if a string is given that is not listed in the structure, the string can not parse as the argument passes to the ***GetValueVector()*** to read the result. Therefore, in the *FieldOpt* commits: "Add new Objective type and complete enum type Property element", we add the three missing "else if and return " statements into the function for both field and well gas injection and well water injection.

4.3.1 Employing extended NPV components to CO_2 case

Given that *FieldOpt* has an optimization example for two wells for well control(example), we modify the well fluid type from Oil-Water to *CO2*-Water and the injection well control type from BHP to Rate. The former target of the model is to find an optimal production well lactation and the profile schedule for both injection and production to give better sweep efficiency from water flooding. We have the similarity for well control but aim to inject as much volume CO_2 as possible versus total pore volume without CO2 production. The result demonstrates that the extended NPV component has extended the *FieldOpt* to optimize a CO_2 case. The case file can be found in the GitHub repository directory: CO2WaterOpt:

We have to comment here that the purpose of modifying this model and performing the optimization is only to check if the *FieldOpt* can optimize a CO_2 case after the extension of NPV component to CO_2. The optimization result will not be investigated. In chapter 5, we have a close case study in which we will both have the application of this extension and the new implementation.

Model Introduction

The model geometry is a number of 12*12 grid cells and a square-shaped reservoir in the horizontal plane with two layers in the vertical direction, shown by Figure 4.3.1. Meanwhile, the initial pore space is full of water, and the injection well is placed in the corner; the production well is located in the diagonal line of that corner, approaching the center of the square model.

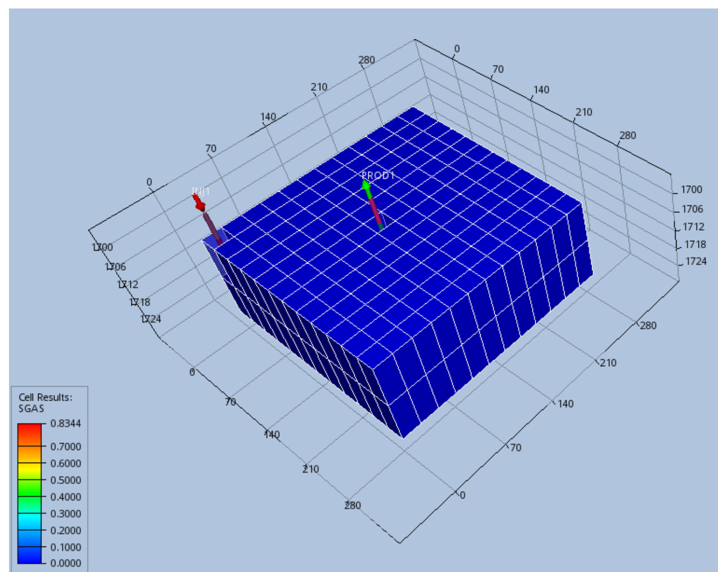

Figure 4.3.1: two dimension Model with two well

We can visually tell from Figure 4.3.2 that the CO_2 breakthrough happened, and there is still a large area that can be flooded by CO_2 after simulation.

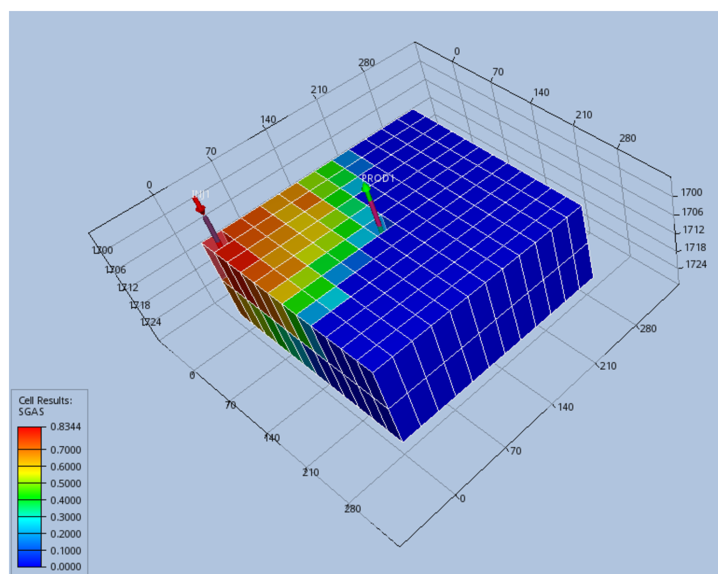

Figure 4.3.2: Simulation Model of the 2D model with two well

FieldOpt **Optimization**

Then, we run FieldOpt with the genetic algorithm (GA) to optimize the same base case. It has the same initial value for two control settings and gives the optimal result respectively; the two methods give a similar setting at six control times as shown in Table 4.3.1. The constraint is the upper and lower boundary of the injection rate and BHP.

Well control optimization							
Case	well control time(days)	0	730	1460	2190	2920	3650
Base case	Injection Rate (sm3/day)	4550	4550	4550	4550	4550	4550
	Production BHP (barsa)	150	150	150	150	150	150
GA optimal case	Injection Rate (sm3/day)	4674	4721	4800	4800	4754	4800
	Production BHP (barsa)	180	80	137	129	180	180

Table 4.3.1: Optimal result for two 2D model

Then, we can visualize the optimal case model, and they all set the production well in the opposite corners of the diagonal line model relative to the injection well location, the Figure 4.3.3 display the optimized CO_2 saturation distribution in the model. The Figure 4.3.4 also shows that the NPV is increasing as the iteration number goes up, which indicates that with the new extension, *FieldOpt* have provided the support for the optimization CO_2 optimization case.

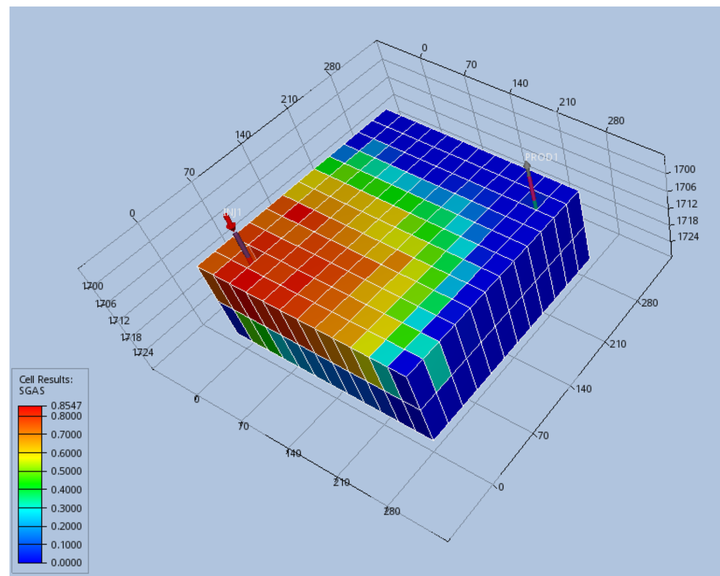

Figure 4.3.3: Simulation of Optimal case for the 2D model

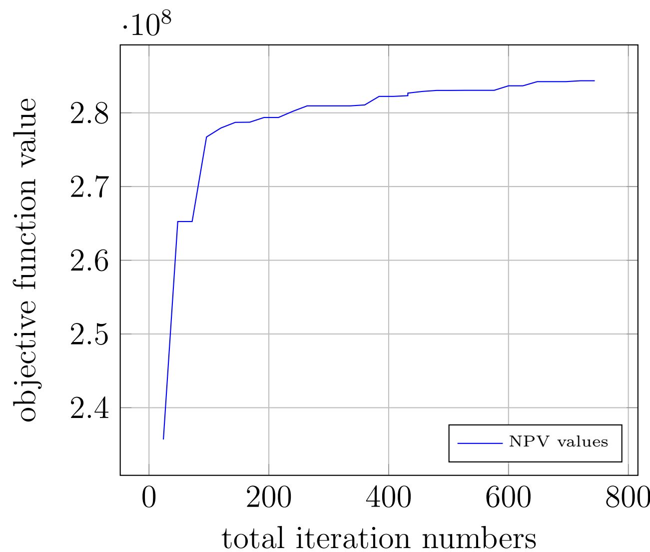

Figure 4.3.4: NPV values versus total interaction numbers plotting

4.4 External objective function calculations

It is preferred that the new objective function be programmed with C++ language, though this choice would maintain the efficient computation performance and integrity of the software. At the same time, the C++ complex syntax, verbose script, and memory management with pointers lead to difficulty in the application of the problem to *FieldOpt*. The form of the objective is often case-specific, as we see from the literature review in chapter 2. As a result, the implementation of a customized objective function requires a solid foundation of programming work and code tests. This also reduces the repeatability of the code in practice.

We, therefore, implement a new objective type - External result - to give full flexibility and applicability to the objective function value calculation. The new method is independent of the existing objective function type. Once the desired type is given in the *FieldOpt* JSON input, where only the selected method executes the calculation task. We design the new ***objective*** namespace as shown in Figure 4.4.1. ***Externalresult*** class to work only as an application programming interface(API) to pass the result to *FieldOpt*. It can be seen in Figure 4.4.1, We have the new derived class ***Externalresult*** showed by green color. it also has the constructor function ***Externalresult()*** that parses the only file path argument into the class. the nested struct class ***Component*** have the function ***passValueFromFile()***, thereby its name, perform the result reading task.

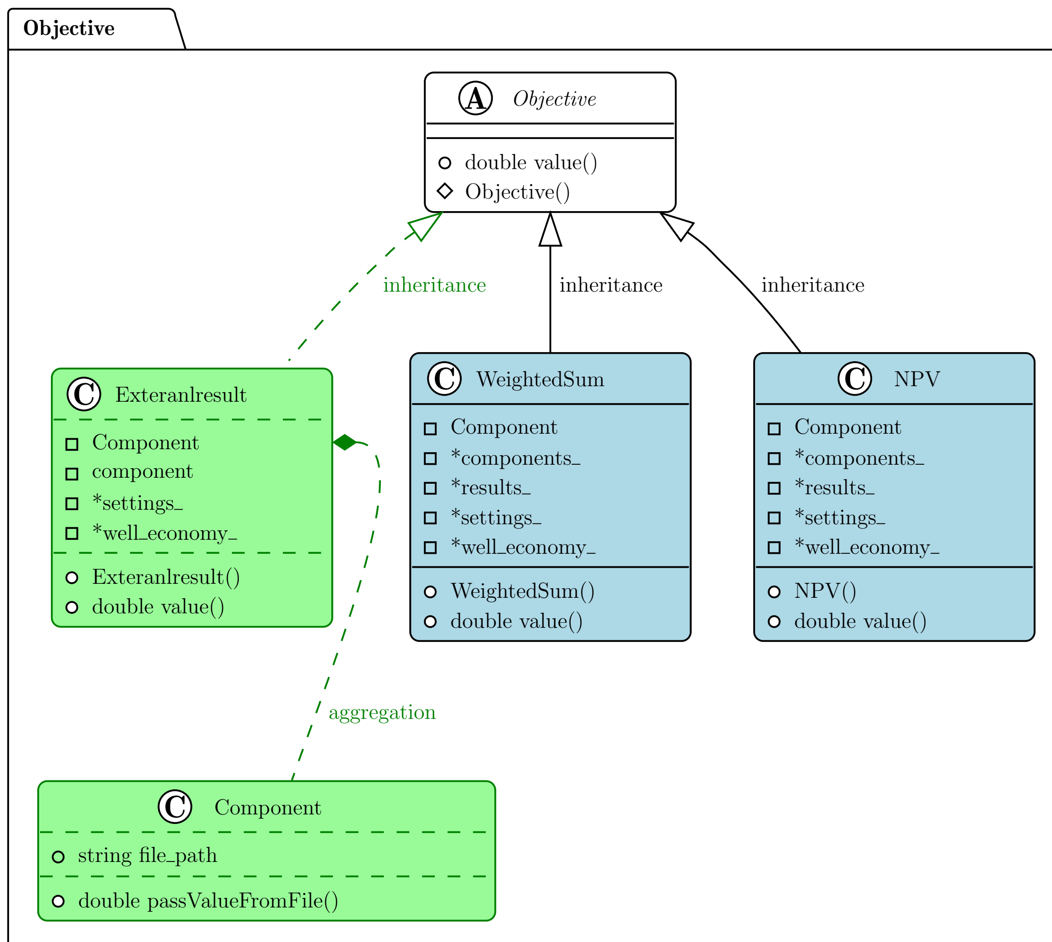

Figure 4.4.1: New objective namespace diagram

Now, let us go to the optimization loop of the designed *FieldOpt* workflow below; we can

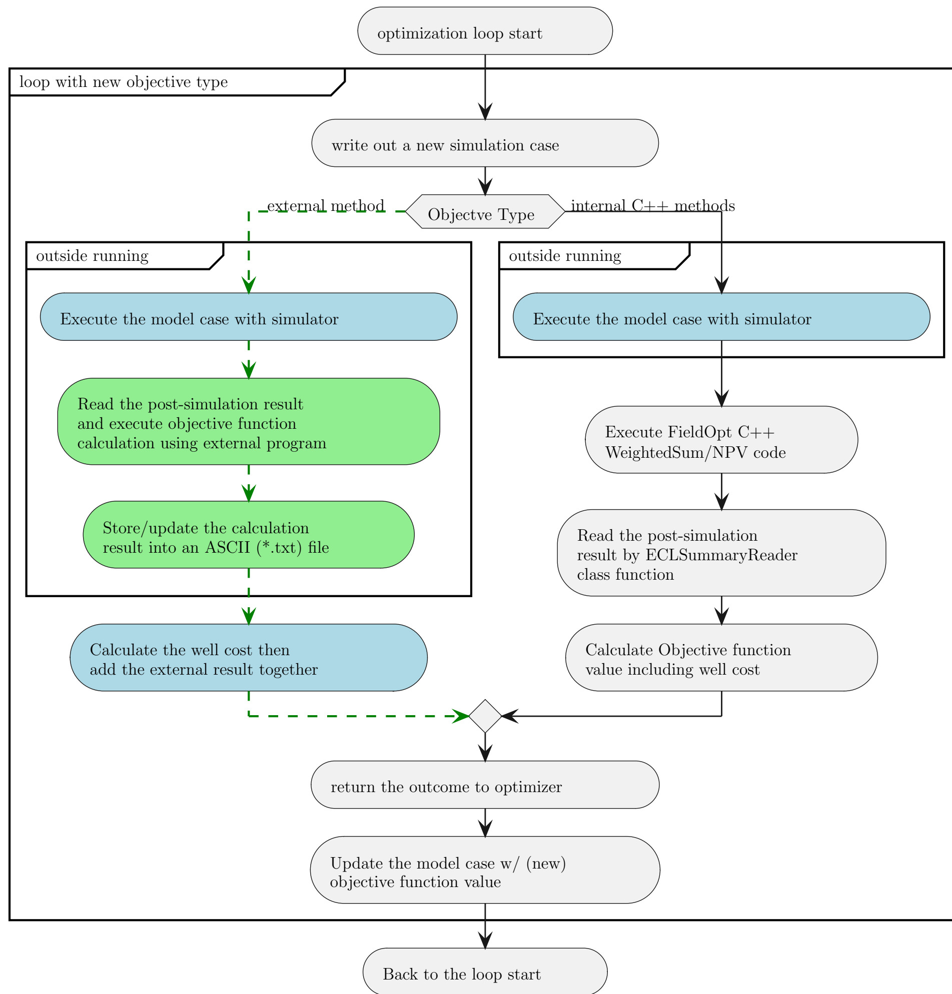

Figure 4.4.2: New implementation feature in the optimization loop

find that the outside running range has been expanded, and the NPV result calculation, storage is added outside the *FieldOpt* running scheme. It has to be noted that the well cost calculation is not a mandatory step in the green dash arrow chart flows. If we specify the well cost calculation is enabled in the JSON input data, the only step that involves the objective function calculation inside the *FieldOpt* running would drop out from the loop and give the full flexibility to the user to the objective function definition and calculation.

CHAPTER
FIVE

CASE STUDY

In this chapter, we will discuss the synthetic model and the initialization of the joint optimization case. The GA and PSO optimization algorithm is selected to maximize the objective function value. The Python interpreter is the tool that we use to calculate the objective function value outside the *FieldOpt* framework; the relevant file and code to this example case can be found at the *FieldOpt* Github repositories: *CO2JointOpt2w*.

5.1 Tilted model

This model is a 10×10×15 grid and tilted to represent a geological structure with a dip. We resized the grid into a small section to reduce the running time and produce the result since we have limitations on the computational force in the personal laptop. As we said before, we use the corner point grid to generate the grid data and the Gaussian covariance model for the reservoir properties. The permeability field is calculated with a multiplier to scale up the porosity field. Strictly speaking, this is not realistic, but at least a study shows that permeability and porosity are positively correlated in sandstone (Nelson 1994). We copy the permeability value from the X to Y dimension, and the Z direction permeability is 0.1 times the value of the X direction. The initial fluid saturation, cell pressure, and CO_2-water ratio, etc, are assigned with a constant value for each cell. The Figure 5.1.1 displays the data pattern in 3D with the same variance of the variable but a different scale for the permeability and porosity.

We deployed two wells for the field operation: a well for water production and an injection well for storing the CO_2 at the reservoir. The reservoir is initially fully saturated with brine, and the production well is fixed in the corner. We put the injection well around the center of the grids. We optimize well control for both wells and conduct well placement optimization only for the injection well. The starting plan is ideally to produce the brine for ten years, and the injection well is shut. After the reservoir pressure has been released, we will simultaneously inject CO_2 and produce water for 50 years. This is a theoretical plan, but in the optimization configuration, we keep the two well states opened all the time and optimized by the *FieldOpt* framework. The injection well sets the BHP as the constraint, the target as the flow rate, and only BHP as the target for the production well; the reason is that it is easier to operate the CO_2 injection rate at the surface with less cost while we have to use the down gauge to monitoring and control equipment for the BHP. In partial, another way could also apply to the project if necessary.

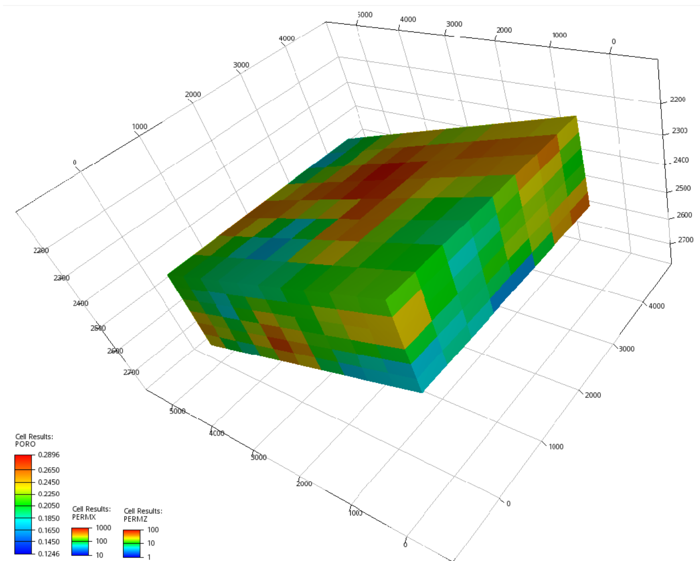

Figure 5.1.1: Random field generation data for the model

Then, we simulate our base case with the simulator *flow*, and we can visually find that much of the reservoir space has not been flooded by CO_2 in Figure 5.1.2. The case is then repeatedly optimized with different objective function types for two algorithms. In the Ketzin pilot site(Brandenburg, Germany) CO_2 injection project, the CO_2 is pumped from the CO_2 intermediate storage tank where the density is about 1,020 kg/m^3 in the -18 °C/21 bar condition, We assume that the surface CO_2 inject density is constant to this value for the injected CO_2 weight approximation. The base case has around 1000 Mt CO_2 injection.

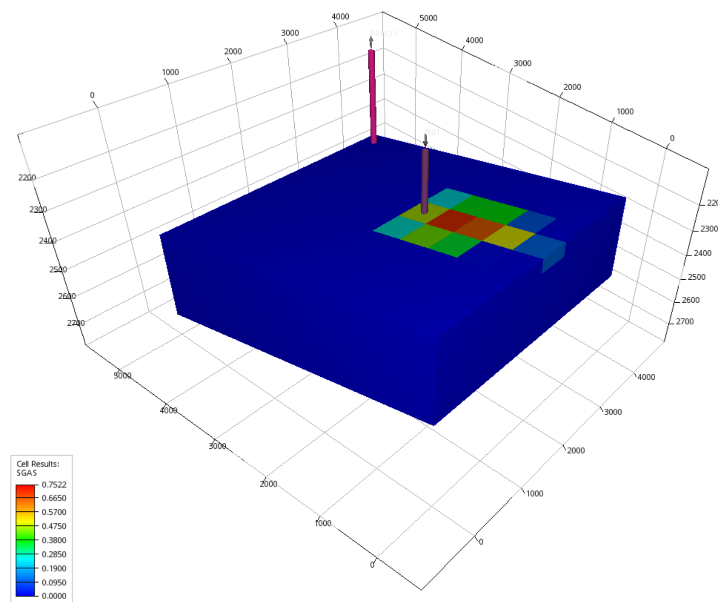

Figure 5.1.2: Base case simulation result

Next, we decided to jointly optimize this CO_2 injection case, initialize the control variable value at the starting time, and define the constraints.

5.1.1 Well control schedule

In this case, we have 60 years to operate the CO_2 injection. We have a set of control times when we will change the production well BHP target and injection well flow rate target. We have no constraint for the production well; the injection well BHP constraint is set to 300 bar all the time, which could be out of consideration for causing high-pressure build-up. Then, we have the boundary for the injection rate ranging from 50 bar to 100 bar. The injection rate ranges from 500000 to 5000000 sm^3/day. The two wells are kept open all the time until the simulation stops. The Table 5.1.1 lists the settings for well control.

Type	Constraint		Control times								
	upper limit	lower limit	0	1825	3650	5475	7300	9125	10950	12774	14600
Injection well rate target(sm^3/day)	500000	5000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000	1000000
production well BHP target(bar)	50	100	100	100	100	100	100	100	100	100	100

Table 5.1.1: well control setting

5.1.2 Well position definition

The well is defined by the toe and heel methods, which are an array of coordinates. the well can either be defined by *WellSpline* or *WellBlock*, *WellBlock* is the array of the integer that is the perforation grid well of the reservoir, while the actual well coordinates are defined by the float array of *WellSpine*. Those two points coordinate is connected to define a line that represents the well trajectory.

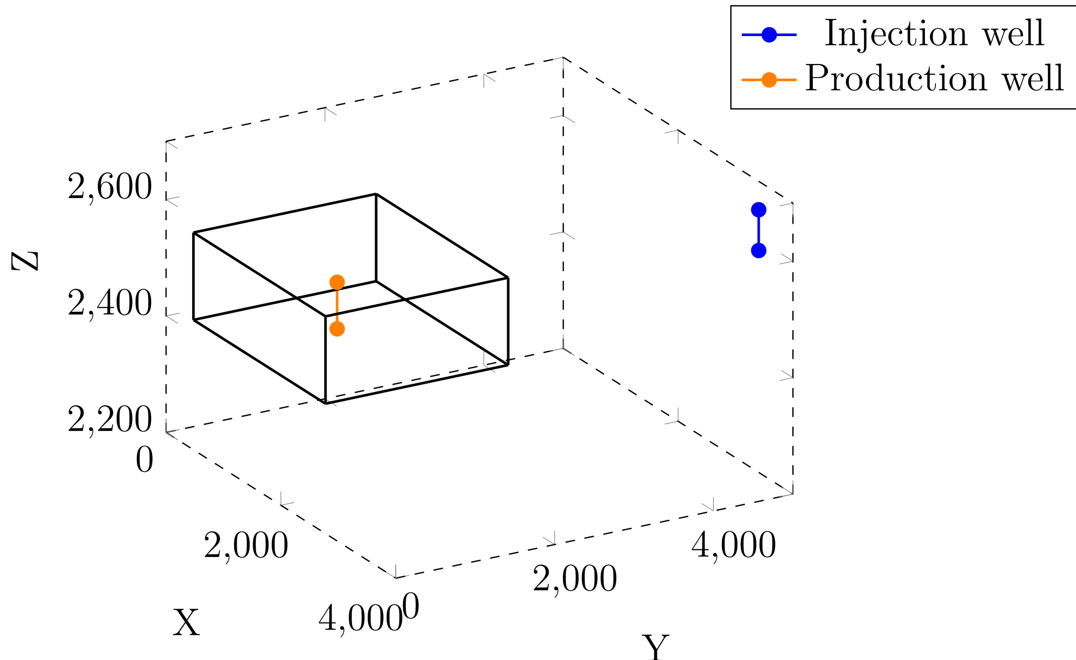

Figure 5.1.3: *WellSpline* coordinates and boundary

The Figure 5.1.3 displays the initial Wellspline coordinates and the boundary for the well injection well in a 3D. The red dot line is the production well completion interval with the reservoir. The orange defines the injection well inside the constraints as a box-shape boundary. The injection well is optimized within this box boundary, and there is also a *WellSpline* length constraint for the injection well.

Type	Injection well Constraint		WellSpline length		Injection well WellSpline coordinates		Production well WellSpline coordinates	
	upper limit	lower limit	Minumu	Maximum	Point 1	Point 2	Point 1	Point 2
X axial	2500	200	50	150	1250	1250	3750	3750
Y axis	2500	200			1250	1250	4750	4750
Z axis	2550	2400			2420	2500	2610	2680

Table 5.1.2: WellSpline setting and constraints

The Table 5.1.2 lists the two wells *WellSpline* coordinates and constraints, all the numbers units is meter in our case.

5.2 Optimization configuration

5.2.1 NPV components

The NPV components usually contain the parameters to calculate the objective function value. In our case, the main income or cost comes from the fluid and well. The component for each fluid includes the fluid price, discount rate, and NPV interval. In Figure 3.1.3, we also have the Boole value that is used for specifying the situation of how the NPV is calculated. We have the field and well-type data that refer to the different contexts of reservoir management. In our predefined code, the field data gives the "cumulative" word in front of the component name, and we have the "well" put in front of the well data. The NPV components have consistent discount rates and fluid prices in the model simulation time, as well as constant time intervals yearly for the NPV calculation. All the data is the field type.

We note that this NPV components setting is identical for both the objective function. We follow the content above to design the tuple variable - a Python-type data structure that takes the same value for each component. The Boole value is not required to be included anymore since we have created the data type that is going to execute the same NPV calculation with our customized Python script. This is also a demonstration that the new external objective function calculation method improves the flexibility and availability of different cases. The well cost inputs, however, are only able to be retrieved from the JSON driver file because we still call the *FieldOpt* internal *model* class function to calculate the well cost.

NPV component	Data type	fluid price [$/sm^3]	discount rate	interval	Well cost	price[$/m]
water production	field	-1e+10	0.08	yearly	drilling cost related length	7500
CO_2 production	field	-10			drilling cost related to XY plane	10000
CO_2 injection	field	15			drilling cost related to horizontal plane	5000

Table 5.2.1: NPV component and well cost setting

We summarize the NPV component value based on those two data structures and list them as Table 5.2.1. The well cost is divided into three types: well cost that is promotional to the well length. As the name suggests, well cost in XY and horizontal planes is the cost of the horizontal plane and cross-section perpendicular to the wellbore. The

last comment is that The fluid price coefficient for CO_2 injection, water production, and CO_2 production is theoretical and hypothetical for producing an intuitive and reliable optimal result. However, the real CO_2 tax credits are not profitable enough to give a considerable economic value. Therefore, the price coefficient requires more detailed investigation, considering unit conversion, data types, etc., to make the result predictive and referable in the practical study.

5.2.2 Methods and penalty

The last section for the optimization running configuration is the algorithm; we select the GA and PSO as the methods because we have various decisive variables in different search ranges. In Table 5.1.1, we note that the search space for the optimal inject rate is much larger than the production target. This makes the compass search not suitable for this case because it has a pre-defined step length. Time steps greater than a hundred would cause the BHP to violate the constraint at the beginning of the optimization, while short time steps less than ten also entail that it might be too slow to reach the global optimum. The GA and PSO could resolve this issue because they are stochastic and population-based methods.

We also applied a strong penalty to the objective function of our case; the CO_2 production cost for the CO_2 is much bigger than the income of the CO_2 injection. This is because we always care much about the CO_2 leakage in the project, and it is not even possible to have this catastrophic situation happen.

5.3 Python scripts

Python is the tool that we are going to use to calculate the objective function value when we choose the external result as the objective type. We put all the related files that involve the objective function into one folder; it contains the the NPV component setting of the example case, a template of the objective function calculation class and a path handler to facilitate running this example case.

5.3.1 Python Objective class

The Python script is written as a prototype for the Equation 3.5 definition; the parent class *ObjFunCla* has two child classes *NPVCal* and *WelCosCla* that are used to calculate the NPV value and well cost value. It could be modified and extended with more instance variables for all the classes. We use a third-party package *resdata* to read the simulation result. We number the key steps that the code performs to calculate the NPV. the source code is available on the examples/Flow/CO2JointOpt2w/PythonFile/ObjFunClass.py.

1. **Initialize the instance variable**

   The input parameter for the class should include the simulation “UNSMRY” result data path and NPV components. The NPV component section data type is a tuple with three dictionaries and is instantiated in the objective case file; the *__init__()* function performs the data initialization work in the parent class.

2. **Reprocess the input parameter**

   The well name, coefficient, discount rate, etc., should be reprocessed in order

to use the *resdata* package for reading simulation results with the corresponding interval. One of the key methods is *numpy_vector*, which takes the keywords and time index as two parameters to return the volume vector data. We have options for both field and well data types. This process is executed by *NPVcompro()* method. An empty list is created and appended to store those variables. The *FieldOpt* NPV C++ code has a similar idea of handling this.

3. **NPV calculation execution**

   The NPV calculation is done by two loops regarding the inner time and outer the NPV component, respectively. This is consistent with the definition of Equation 3.4. The *NPVsumcal()* function makes the final calculation on the condition of completing the NPV component variable processing. In this case, we use the *FieldOpt* internal well cost result, adding to the NPV calculation to the external methods. While it is open to define the other objective function terms like well cost as a child class of

5.3.2 Python Objective case

This Python case file contains a tuple data structure - a variable to store the NPV component argument, such as well name, data type, fluid price, etc. This Python file imports the ***Objective*** Python class in order to repeatedly calculate the objective function results after the finish of the execution of the simulation command in the running program. The Python file takes the optimized case UNSMARY file path and external objective result file path as arguments and stores the values into an ASCII (*.txt) file. Finally, it returns or updates the value in the target file that is going to pass to *FieldOpt*.

5.3.3 Path handler

The path handler is a Python script designed for running facilitation. The Python script file takes two arguments as input: the JSON driver file path and the (optimization) output directory path. It mainly works to create the objective function result file in the output directory and pass the external objective function result file path argument into the JSON drive file. This file is going to be executed only once by the *run_example_Python*. After the file path argument is assigned to the JSON driver and an ASCII (*.txt) file is created, the *FieldOpt* stores the path and updates the result by the objective case file. In fact, the path handler originally takes the file path arguments from the extension of the present work directory of *run_example_Python*.

5.4 Shell script

The shell script is the command that executes the program running outside the *FieldOpt*; here, the simulation execution, the following NPV calculation with Python interpreter, is responsible for the *base_bash* file with a newly added command. The new bash script is repeatedly executed in the optimization run.

The results store and file path argument passing to the JSON and *FieldOpt* are completed sequentially by the *run_example_Python* bash script. We also add a command to run the base case model only once to the *run_example_Python* because the *FieldOpt* requires the input of the grid file, which is generated after the base case simulation run.

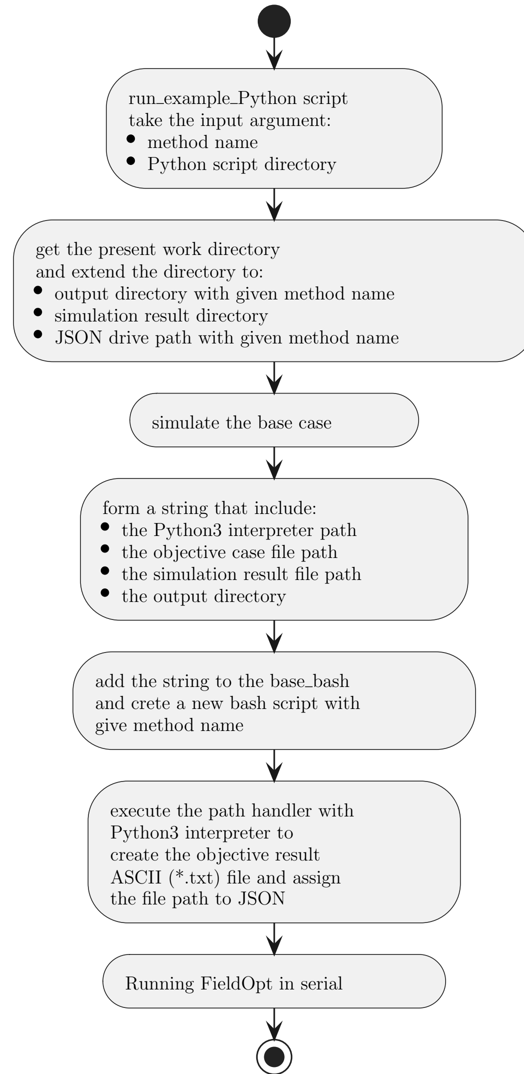

Figure 5.4.1: Shell script command workflow for running facilitation

The Figure 5.4.1 above explains the process that the bash script does sequentially to run the simulation case. The bash shell demand can be found in the README.md file in the example folder of the repository.

5.5 Outcome

After we run the simulation with *FieldOpt*, we get optimization results. The GA algorithm has the error that it can not calculate the well index for the external result calculation. We, therefore, only have 3 data sets in our result.

Type	Internal NPV type		External
	GA	PSO	PSO
objective function value($)	2.814582e+10	2.69769e+10	2.133529e+10
Injected CO2(sm3)	1.40978e+11	5.25906e+09	4.50088e+09
produced water(sm3)	5.10955e+8	4.17748e+8	4.59161e+07

Table 5.5.1: Optimization result from FieldOpt.

From the Table 5.5.1, if we take the center of total injected CO_2, we can see that the best case is the genetic algorithm calculated by the internal methods. For each case, we do not have CO_2 production in all the our there case. The simulation of the result can be seen from the Figure 5.5.1.

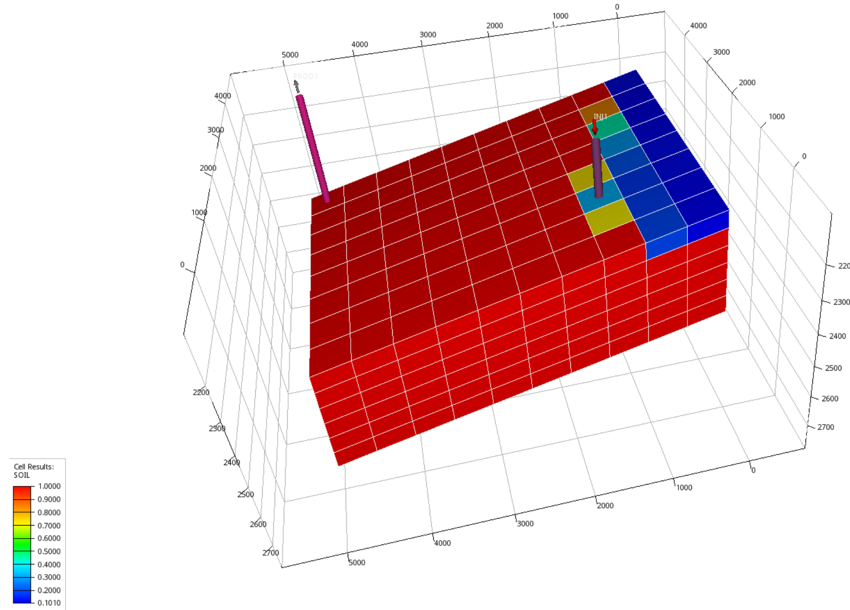

Figure 5.5.1: Best case simulation result

Lastly, we plot the three optimization case objective function with respect to time, shown by Figure 5.5.2

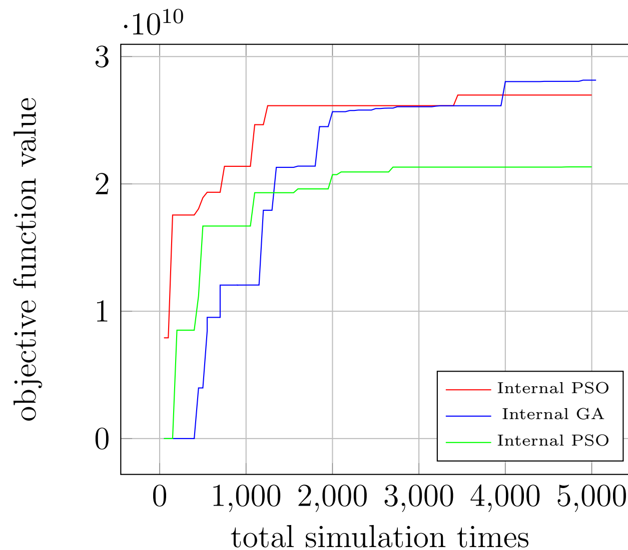

Figure 5.5.2: *FieldOpt* internal and external calculation in three optimal cases

We can see from the plotting that both the internal and external algorithms are optimizing the case. The PSO algorithm optimizes the case in a few iterations in the beginning and gradually approaches the optimum, while the GA algorithm takes long steps but overtakes the PSO after 5000 iterations. More optimization should be run when parallel computing is available to produce more outcomes and investigate the result further.

CHAPTER

SIX

SUMMARY AND FUTURE WORK

6.1 Summary

CO_2 storage is an essential topic and means in the context of the global warming challenge. Application of CO_2 storage is challenging as we have limited research and projects compared to the study of the traditional petroleum industry over hundreds of years. However, the experience, techniques, and technology that we have accumulated during the era of hydrocarbon production give support to the development of CO_2 sequestration projects. As an example, most of the reservoir simulation software has developed new features to support the simulation of CO_2 flow.

As we already have an optimization tool, *FieldOpt*, that has implemented many features needed to optimize field development management, it is natural to update the *FieldOpt* code base to facilitate the study of CO_2 sequestration. This master project work makes the current software available for CO_2 sequestration. This includes extending the objective class in *FieldOpt* to be able to handle gas injection as an income in the objective function calculations. It also includes an implementation of a flexible API to conduct the objective function calculation outside of the *FieldOpt* framework. The latter is considered essential for CO_2 sequestration, as the NPV calculations there are often fit-for-purpose, with very specific pressure conditions and penalties when CO_2 flows to certain regions of the reservoir.

In this thesis we have also tested the new implementations on a purpose-built problem case including a test model of a simple tilted reservoir. We have optimized our problem with a total of three runs for each tested optimization algorithm to be able to compare the development. The results show that both the algorithms and the internal and external objective function calculation work well to optimize our CO_2 storage problems.

6.2 Future work

A master's thesis is very limited in time, there are therefore several topics that should be considered further. These include the following topics.

6.2.1 Improve the case study

The developed simple reservoir model can be developed into a more realistic case for sequestration modeling. The optimization result could probably be significantly improved because we see that there is still a large unused space for CO_2 storage, and the optimal cases tend to produce CO_2. The first suggestion is to increase the strength of the CO_2 production penalty to limit CO_2 production. The second suggestion, the most important one, is to adjust the constraints and initial decisive variable to confine the optimization to a better search region.

6.2.2 Diverse objective function

We give the prototype of the Python objective class in our example case, and it could be extended with more child classes. We use the CO_2 production as the penalty, but we still have other choices, such as the dissolved CO_2 - brine ratio Rs or CO_2 saturation SGAS. We could also introduce a proper penalty weight on either Rs or SGAS for the production well completion block cell. The well cost could be discounted every year if we applied a statistical relationship between the rate of penetration (ROP) versus time in a depleted hydrocarbon reservoir. Penalties for pressure increase in the reservoir, particularly towards the overburden, could also be implemented.

6.2.3 Parallel computing

When we designed our ***Extrenalresult*** class, there was only one file that stored the objective value. We serially run our case on a personal laptop, and in this setting the program works successfully and records every case objective function value. However, we have not tested that all the external result values would be stored and read correctly in a parallel computing setting. A possible option is to apply the universally unique identifier(UUID) mechanism to each objective function value. The UUID technique has already been implemented in the *FieldOpt* to variable values (Baumann, Dale, and Bellout 2020). Therefore, this module can be reused also for the external objective function calculations. The external result class is an interface to pass the external result, but the value could be stored with the UUID method and read by *FieldOpt*. Such a new function will then replace the simpler ***readValueFromFile()*** function created in this thesis.

REFERENCES

Anthonsen, Karen L et al. (n.d.). "This memo describe technical terms and concepts used in relation to CO2 storage. This text is included in the Nordic CO2 Storage Atlas availeble at :" in: ().

Vishal, V. and T.N. Singh, eds. (2016). *Geologic Carbon Sequestration*. Cham: Springer International Publishing. ISBN: 978-3-319-27017-3 978-3-319-27019-7. DOI: 10.1007/978-3-319-27019-7. URL: http://link.springer.com/10.1007/978-3-319-27019-7 (visited on 06/05/2024).

Ringrose, Philip and Mark Bentley (2021). *Reservoir Model Design: A Practitioner's Guide*. Cham: Springer International Publishing. ISBN: 978-3-030-70162-8 978-3-030-70163-5. DOI: 10.1007/978-3-030-70163-5. URL: https://link.springer.com/10.1007/978-3-030-70163-5 (visited on 06/06/2024).

Müller, Sebastian et al. (Apr. 12, 2022). "GSTools v1.3: a toolbox for geostatistical modelling in Python". In: *Geosci. Model Dev.* 15.7, pp. 3161–3182. ISSN: 1991-9603. DOI: 10.5194/gmd-15-3161-2022. URL: https://gmd.copernicus.org/articles/15/3161/2022/ (visited on 06/16/2024).

Cameron, David A. and Louis J. Durlofsky (Sept. 1, 2012). "Optimization of well placement, CO2 injection rates, and brine cycling for geological carbon sequestration". In: *International Journal of Greenhouse Gas Control* 10, pp. 100–112. ISSN: 1750-5836. DOI: 10.1016/j.ijggc.2012.06.003. URL: https://www.sciencedirect.com/science/article/pii/S1750583612001296 (visited on 06/22/2024).

Ampomah, William et al. (2017). "Co-optimization of CO2-EOR and storage processes in mature oil reservoirs". In: *Greenhouse Gases: Science and Technology* 7.1. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ghg.1618, pp. 128–142. ISSN: 2152-3878. DOI: 10.1002/ghg.1618. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/ghg.1618 (visited on 06/26/2024).

Ringrose, Philip, Sallie Greenberg, et al. (July 1, 2017). "Building Confidence in CO2 Storage Using Reference Datasets from Demonstration Projects". In: *Energy Procedia*. 13th International Conference on Greenhouse Gas Control Technologies, GHGT-13, 14-18 November 2016, Lausanne, Switzerland 114, pp. 3547–3557. ISSN: 1876-6102. DOI: 10.1016/j.egypro.2017.03.1484. URL: https://www.sciencedirect.com/science/article/pii/S1876610217316788 (visited on 06/21/2024).

Baxendale, David (n.d.). "OPEN POROUS MEDIA". In: ().

Rasmussen, Atgeirr Flø et al. (Jan. 1, 2021). "The Open Porous Media Flow reservoir simulator". In: *Computers & Mathematics with Applications*. Devel-

opment and Application of Open-source Software for Problems with Numerical PDEs 81, pp. 159–185. ISSN: 0898-1221. DOI: 10.1016/j.camwa.2020.05.014. URL: https://www.sciencedirect.com/science/article/pii/S0898122120302182 (visited on 06/23/2024).

Rykkelid, Karoline Louise (2016). "Graphical User Interface for Petroleum Field Optimization Software (FieldOpt) - Development of Interface for Configuring Driver Files". Accepted: 2017-03-10T14:53:42Z. Master thesis. NTNU. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2433688 (visited on 06/26/2024).

Baumann, Einar J. M., Stein I. Dale, and Mathias C. Bellout (Feb. 1, 2020). "FieldOpt: A powerful and effective programming framework tailored for field development optimization". In: *Computers & Geosciences* 135, p. 104379. ISSN: 0098-3004. DOI: 10.1016/j.cageo.2019.104379. URL: https://www.sciencedirect.com/science/article/pii/S0098300419301013 (visited on 04/07/2024).

Panahli, Chingiz (2017). "Implementation of Particle Swarm Optimization Algorithm within FieldOpt Optimization Framework - Application of the algorithm to well placement optimization". Accepted: 2017-09-04T14:05:09Z. Master thesis. NTNU. URL: https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2453090 (visited on 04/07/2024).

Nelson, Philip H (1994). "Permeability-Porosity Relationships in Sedimentary Rocks". In.

APPENDICES

A - GITHUB REPOSITORY

All code used in this document is included in the Github repository linked below. The README file provides further explanations.

A part of the master thesis work is put on the personal repository:

master-appendix `https://github.com/wangh03/master-appendix`

Appendix A - Github repository link

- ***FieldOpt*** `https://github.com/PetroleumCyberneticsGroup/FieldOpt`
- ***Geostatistical toolbox*** `https://github.com/GeoStat-Framework/GSTools`
- ***ResData*** `https://github.com/equinor/resdata`
- ***Ensemble-based Reservoir Tool*** `https://github.com/Ensembles/ert`

Appendix B - Nomenclature

B.1. symbols for CO_2 injectivity

q_{CO_2}	CO_2 injection rate.
p_{fbhp}	flowing bottom hole pressure.
p_{res}	far-field reservoir pressure.
μ_g	gas viscosity.
Z	compressibility factor of gas.
p	pressure at the given depth.
k_{res}	reservoir permeability.
h_i	the height of the injection/completion interval.
r_e	effective radius of the reservoir unit
r_w	radius of the well

B.2. symbols for the black-oil model equation

ϕ_{ref}	reference porosity
A_α	accumulation terms, the change of component mass over a time step.
$\mathbf{u}_\alpha$	velocity of component α, including the terms of the Darcy's law.
q_α	well outflux density of pseudo component α, depending on the well model

B.3. symbols for discrete equation

$R_{\alpha,i}$	residual of cell i for component α
Δt	time step length for the current Euler step.
$C(i)$	the set of cells connected to cell i.
$u_{\alpha,ij}$	upstream cell for phase α for the connection between cells i and j

V_i	cell volume of cell i

B.3. symbols for NPV definition

N	the total number of the component
$V_{i,t}$	cumulative fluid volume of the component i in time interval t
P_i	fluid price per volume unit for component i
t	the t-th time interval
T	cumulative operational time
r_i	the discount rate for component i
i	the i-th component

Appendix C - the C++ source code

```
std::vector<double> ECLResults::GetValueVector(Results::Property
    prop)
{
    if (!isAvailable()) throw ResultsNotAvailableException();
    switch (prop) {
        case CumulativeOilProduction:   return summary_reader_->
    fopt();
        case CumulativeGasProduction:   return summary_reader_->
    fgpt();
        case CumulativeWaterProduction: return summary_reader_->
    fwpt();
        case CumulativeWaterInjection:  return summary_reader_->
    fwit();
        case CumulativeGasInjection:    return summary_reader_->
    fgit();
        case Time:                      return summary_reader_->
    time();
        default: throw std::runtime_error("In ECLResults: The
    requested property is not a field or misc property.");
    }
}

std::vector<double> ECLResults::GetValueVector(Results::Property
    prop, QString well_name) {
    if (!isAvailable()) throw ResultsNotAvailableException();
    switch (prop) {
        case CumulativeWellOilProduction:   return summary_reader_
    ->wopt(well_name.toStdString());
        case CumulativeWellGasProduction:   return summary_reader_
    ->wgpt(well_name.toStdString());
        case CumulativeWellWaterProduction: return summary_reader_
    ->wwpt(well_name.toStdString());
        case CumulativeWellWaterInjection:  return summary_reader_
    ->wwit(well_name.toStdString());
        case CumulativeWellGasInjection:    return summary_reader_
    ->wgit(well_name.toStdString());
        default: throw std::runtime_error("In ECLResults: The
    requested property is not a well property.");
    }
}
```

Listing 1: Code block in the eclresult.cpp

```
enum Property {
    CumulativeOilProduction,
    CumulativeGasProduction,
    CumulativeWaterProduction,
    CumulativeWaterInjection,
    CumulativeGasInjection,
    CumulativeWellOilProduction,
    CumulativeWellGasProduction,
    CumulativeWellWaterProduction,
    CumulativeWellWaterInjection,
    CumulativeWellGasInjection,
Time
};

Property GetPropertyKeyFromString(QString prop) {
    if (prop == "CumulativeOilProduction") return
   CumulativeOilProduction;
    else if (prop == "CumulativeGasProduction") return
   CumulativeGasProduction;
    else if (prop == "CumulativeWaterProduction") return
   CumulativeWaterProduction;
    else if (prop == "CumulativeWaterInjection") return
   CumulativeWaterInjection;
    else if (prop == "CumulativeWellOilProduction") return
   CumulativeWellOilProduction;
    else if (prop == "CumulativeWellGasProduction") return
   CumulativeWellGasProduction;
    else if (prop == "CumulativeWellWaterProduction") return
   CumulativeWellWaterProduction;
    else if (prop == "Time") return Time;
    else throw ResultPropertyKeyDoesNotExistException("");
}
```

Listing 2: Code block in the result.h