Algorithm for Optimal Well Placement in Geothermal Systems Based on TOUGH2 models

Dagur Helgason

Thesis of 60 ECTS credits
Master of Science (M.Sc.) in Sustainable Energy Engineering

January 2017

Algorithm for Optimal Well Placement in Geothermal Systems Based on TOUGH2 models

by

Dagur Helgason

Thesis of 60 ECTS creditssubmitted to the School of Science and Engineering
at Reykjavík University in partial fulfillment
of the requirements for the degree of
Master of Science (M.Sc.) in Sustainable Energy Engineering

January 2017

Algorithm for Optimal Well Placement in Geothermal Systems Based on TOUGH2 models

Dagur Helgason

January 2017

Abstract

In a world of ever increasing use of renewables geothermal has lagged behind and has seen little growth compared to other renewables due in part to its high capital cost. Geothermal wells account for about a third of the capital cost and it is therefore important to ensure the highest possible success rate and value creation from these wells. In order to address this, an algorithm has been developed that utilizes a numerical TOUGH2 model of a geothermal system to evaluate the optimal well placement based on a net present value estimation. The algorithm was tested using a hypothetical model and found the optimal wells to be in the hottest parts of the model at depth and in the upper heat zone, directly above the heat source in both cases. The algorithm was also subjected to a sensitivity and processing time analysis with the hypothetical model as an input model. The sensitivity analysis showed that the models results were most sensitive to changes in reinjection enthalpy, discount rate and the power plants thermal efficiency. The processing time analysis showed that the algorithm can potentially be run in a reasonable enough time to serve as a tool for decision making.

Keywords: Geothermal, TOUGH2, optimization algorithm, well placement, reservoir engineering

Algoriþmi til Bestunar Borholustaðsetningar í Jarðhitakerfum Byggt á TOUGH2 Líkönum

Dagur Helgason

janúar 2017

Útdráttur

Í heimi þar sem notkun endurnýjanlegra orkugjafa er sífellt að aukast hefur jarðvarmi dregist aftur úr og ekki náð jafn örum vexti og margir aðrir endurnýjanlegir orkugjafar að hluta til vegna mikils stofnkostnaðar. Borholur í jarðhitakerfum eru um þriðjungur þessa kostnaðar, það er því mikilvægt að tryggja hæstu mögulega árangurstíðni af borun þessara hola. Til þess að takast á við þetta var þróaður algoriþmi sem notfærir sér töluleg líkön af jarðhitakerfum fyrir TOUGH2 til að finna bestu borholu staðsetningu innan kerfisins útfrá mati á núvirði hverrar borholu. Algoriþminn var prófaður á fræðilegu líkani af jarðhitakerfi, þar sem hann staðsetti bestu holurnar í því kerfi í heitustu pörtum líkansins beint yfir varmauppsprettunni, annarsvegar á miklu dýpi við varma uppsprettuna og hinsvegar í efra varmasvæði kerfisins. Næmnigreining og mælingar á keyrslutíma voru einnig framkvæmdar á algoriþmanum með þessu líkani. Næmnigreiningin leiddi í ljós að niðurstöður líkansins væru næmastar fyrir framleiðslu stuðli/massaflæði fyrir borholurnar og vermi niðurdælingarvökva. Mælingar á keyrslutíma leiddu í ljós að algoriþminn getur keyrt innan raunhæfra tímamarka og gæti orðið gagnlegt verkfæri fyrir ákvörðunartöku.

Lykilorð: Jarðhiti, TOUGH2, bestunar algoriþmi, borholustaðsetning, forðafræði

Algorithm for Optimal Well Placement in Geothermal Systems Based on TOUGH2 models

Dagur Helgason

Thesis of 60 ECTS credits submitted to the School of Science and Engineering
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science (M.Sc.) in Sustainable Energy Engineering

January 2017

Student:

....
Dagur Helgason

Supervisor:

....
Ágúst Valfells

....
Egill Júlíusson

Examiner:

....
Lárus Þorvaldsson

Acknowledgements

> ...if you trust in yourself... and believe in your dreams... and follow your star... you'll still get beaten by people who spent their time working hard and learning things and weren't so lazy.
>
> Terry Pratchett [1]

I would first like to thank my thesis advisors Dr. Ágúst Valfells of the Iceland School of Energy at Reykjavík University and Dr. Egill Júlíusson of National Power Company of Iceland (Landsvirkjun) without their input and help none of this would have happened. I would also like to thank our research group ORSM, our biweekly meetings kept me on track and your input and feedback helped greatly along the way. I owe Iceland GeoSurvey thanks for supplying me with data to use along the way and Dr. Magnús Þór Jónsson of the School of Engineering and Natural Sciences at the University of Iceland for his help with the joy that is TOUGH2. The last academic I would like to thank is Dr. Adrian Croucher of the Department of Engineering Science at the University of Auckland, his work on PyTOUGH made this thesis doable. I would like to express my profound gratitude to my parents, without who's support I would not have a single degree to my name, my siblings Birta and Breki for showing a varying degree of interest in my work and allowing me to sometimes talk at them despite not understanding a single word. My cats Lilith, Lucifer and Harvey for being an endless source of hilarity. Finally, Helga my girlfriend for putting up with me and supporting me in my studies despite my sometimes less than ideal attitude at the end of a long day. I love you all, thank you.

Preface

This dissertation is original work by the author, Dagur Helgason.

Contents

Acknowledgements **xi**

Preface **xiii**

Contents **xv**

List of Figures **xvii**

List of Tables **xix**

List of Abbreviations **xxi**

List of Units and Variables **xxiii**

1 Introduction **1**
- 1.1 How are wells placed? 1
- 1.2 Background and scope 3
- 1.3 Literature review 3
- 1.4 How can reservoir simulators be used to assist with well placement? 4

2 Methods **5**
- 2.1 Tools used 5
  - 2.1.1 TOUGH2 5
  - 2.1.2 Python 2.7 6
  - 2.1.3 PyTOUGH 6
  - 2.1.4 PetraSim 6
- 2.2 Algorithm structure 6
  - 2.2.1 Inputs 7
  - 2.2.2 Well types 9
  - 2.2.3 Assumptions 10
  - 2.2.4 Generator 10
  - 2.2.5 Batch file 11
  - 2.2.6 PreProcessor 11
  - 2.2.7 Processor 12
  - 2.2.8 PostProcessor 12
- 2.3 Testing 13
  - 2.3.1 Hypothetical model 13
  - 2.3.2 Running the Algorithm 15
  - 2.3.3 Sensetivity analysis 16
  - 2.3.4 Processing time 17

3 Results **19**
3.1 Optimal well placement 19
3.1.1 Well on deliverability 19
3.1.2 Well on wellbore file 20
3.2 Sensetivity analysis 21
3.2.1 Well on deliverability 21
3.2.2 Well on wellbore file 22
3.3 Processing time 23

4 Discussion **25**
4.1 Most important parameters 25
4.2 Dependency on data reliability 25
4.3 Summary 25
4.4 Conclusion 26
4.5 Future work 26

Bibliography **29**

A Code **31**

B Computer Specifications **43**

List of Figures

1.1 Drilling success rate by project phase and decade. The exploration phase includes the first 6 wells drilled, the development phase includes well 7 through 30 and the operation phase includes all wells after the 30th.[6] 2

2.1 Flowchart describing how the Algorithm runs. The further to the right an operation is the later it is run. The Inputs are supplied by the user and each is described in Table 1. The user should run Generator.py which generates a PreProcessor.py script, a user defined number (N) of ProcessorXX.py scripts, a PostProcessor.py script and a batch file. The batch file makes sure the scripts are run in the correct order, first the PreProcessor.py, followed by all the ProcessorXX.py scripts and the PostProcessor.py script. PreProcessor.py creates the Input files needed for the simulations, these are sorted into N Input batches that are used as inputs for each of the ProcessorXX.py scripts. The Processor.py scripts each go into a simulation batch and run every Input file there creating N Output batches that contain the well data needed for the final step. The PostProcessor.py goes through the output files, evaluating the NPV of each well locations, and returns a file that lists all the simulated wells in order of their NPV, displaying their NPV, name and location coordinates within the model. 7

2.2 Visual representation of the Hypothetical model, used for testing the Algorithms, showing the three different rock types and their locations within the model. The top layers are the cap rock represented with a brown color, the reservoir rock is represented with a blue color and finally the bottom layers consist of the base rock represented with a red color. Two corner points are shown on the figure as well as their coordinates, these are at the bottom the point (0.0; 0.0; 0.0) and at the top the point (10,000.0; 10,000.0; 10,000.0) 14

2.3 Plane slices showing the temperature of the Hypothetical model at steady state. 15

2.4 Well bottom pressure as a function of mass flow and enthalpy for well BJ14 in the Bjarnarflag geothermal field[20]. 16

3.1 First 12 lines of the output file from the Algorithm in the well on deliverability case. 20

3.2 First 12 lines of the output file from the Algorithm in the well on wellbore file case. 20

3.3 Results of sensitivity analysis of the well on deliverability case. Each input was increased by 10% and the change in the highest NPV as well as the number of positive NPVs was measured. 21

3.4 Results of sensitivity analysis of the well on deliverability case. Each input was increased by 10% and the change in the highest NPV as well as the number of positive NPVs was measured. 22

3.5 Results of the processing time analysis done using the Hypothetical model with the Algorithm. Results are shown for both the well on deliverability and well on wellbore file cases along with a fitted trendline and its equation. Note that the trendlines are fitted to the logarithmic data and not the original data and the x value has to be the logarithm of the number of cells and will give the logarithm of the processing time as a result. 24

B.1 Hardware specifications of the computer used to test the Algorithm. 43

List of Tables

2.1 Description of the Algorithms inputs. 9
2.2 Properties of the three rock types in the Hypothetical reservoir model used for testing the Algorithm. 13

3.1 Cell name, NPV, XY-coordinates and the depth from surface for the top five best wells in the Hypothetical model using the well on deliverability setup. 19
3.2 Cell name, NPV, XY-coordinates and the depth from surface for the top five best wells in the Hypothetical model using the well on wellbore file setup. 20

List of Abbreviations

PV	Photovoltaic
CSP	Concentrated solar power
LCOE	Levelized cost of electricity
EIA	U.S. Energy Information Administration
IFC	International Finance Corporation
MT	Magnetotelluric
TEM	Transient-Electromagnetic
ORSM	Operations Research for Subsurface Modeling
MARS	Multivariate adaptive regression spline
NPV	Net present value
EOS	Equation of state
EOS1	Equation of state 1. Water, water with tracer
O&M	Operation and maintenance
RMS	Root mean square

List of Units and Variables

Units	
%	Percentage
$	Dollars
W	Watt
kW	Kilowatt
MW	Megawatt
MWh	Megawatthour
MWe	Megawatts electric
m	Meters
kg	Kilogram
s	Second
J	Joule
kJ	Kilojoule
K	Kelvin
°C	Degrees centigrade
Pa	Pascal
bar-a	Bar absolute

Variables	
$Revenues_i$	Revenues in year i, in $
$Costs_i$	Costs in year i, in $
z	Depth, in m
r	Discount rate, a number between 0 and 1
$Price$	Price of power to operator, in $/MWh
$\dot{m}_i$	Mass flow rate, in kg/s
h_i	Enthalpy of geofluid in year i, in kJ/kg
h_0	Enthalpy of reinjected geofluid, in kJ/kg
η	Thermal efficiency of power plant, a number between 0 and 1

Chapter 1

Introduction

Renewable energy in the world is growing at a faster pace than before and average growth rate of renewable power production (includes Geothermal power, hydro-power, Solar PV, CSP and Wind) in 2015 was around 12%. Even with this increased focus on renewables geothermal power is growing slowly and had a growth rate of only 2.4% in 2015[2]. A variety of factors contribute to this low growth rate one of them being the high capital cost of geothermal plants. While the Levelized Cost of Electricity(LCOE) for geothermal power is the lowest out of all technologies according to the EIA[3] ,for plants coming online in 2022, the capital cost is relatively high at a $6,230/kW for a dual flash geothermal power plant in 2012 compared to for example onshore wind at $2,213/kW and large scale photovoltaic at $3,873/kW [4]. The majority of this capital cost comes from the construction of the power plant itself and the drilling, in Iceland the construction of the power plant accounts for approximately 35% of the investment cost and the drilling for approximately 34%[5].
With this high cost of drilling wells, it is important that as many wells as possible are successful and that they be as profitable as possible. The IFC defines a successful well, in short, as a well that produces more than 3MW$_e$, in the case of production wells, or have been active for a long time, in the case of injection wells [6]. According to the IFC the success rate of the first well in a field is approximately 50% and the cumulative success rate at the end of the development phase (when 30 wells have been drilled) is approximately 70%. The same report showed the drilling success rate over time, by project phase for the past five decades, which showed that a steady increase in success rate has taken place in the exploration phase (defined as the first 6 wells drilled) while the success rate in the development phase (defined as wells 7 to 30) and the operation phase (defined as all wells after the 30th well) did not change significantly, see Figure 1.1. The figure shows that in the 2000s the success rate of exploration wells surpassed the success rate of wells in the development phase which would indicate that success rate decreased as more knowledge about the geothermal systems in question was acquired.
If success rate can be increased in geothermal drilling the results would be decreased capital cost as fewer wells would need to be drilled for a given power output. On top of that if the wells that are drilled are more profitable investors are more easily attracted.

1.1 How are wells placed?

Many companies have guidelines and requirements that have to be met before a well can be placed. At the time of writing, late 2016, Landsvirkjun (the National Power Company of Iceland) was working on these guidelines, for their operations, and access to a draft of those

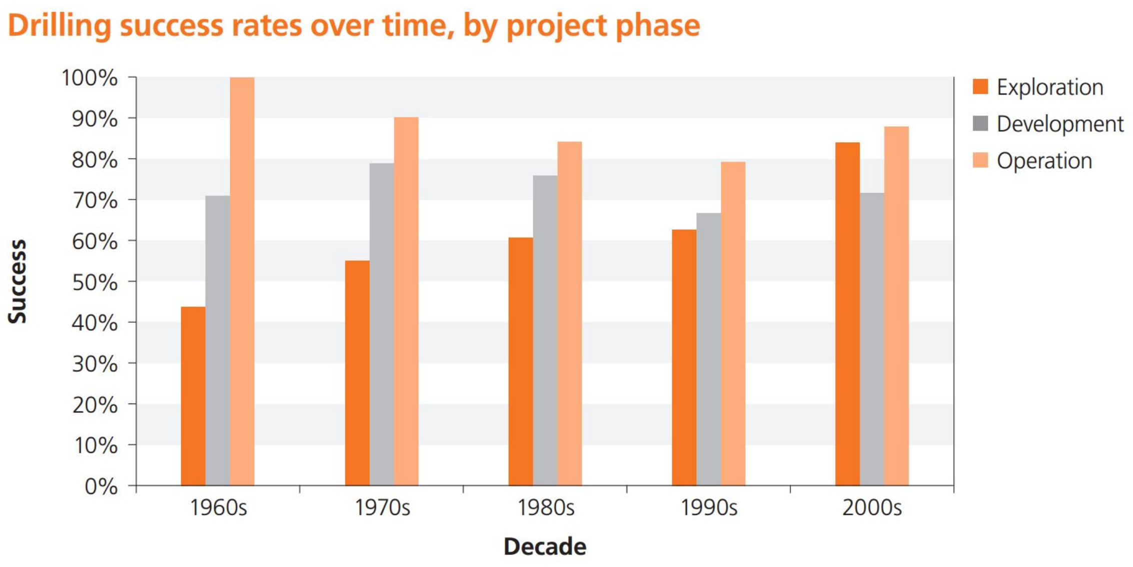


Figure 1.1: Drilling success rate by project phase and decade. The exploration phase includes the first 6 wells drilled, the development phase includes well 7 through 30 and the operation phase includes all wells after the 30th.[6]

guidelines was granted for the writing of this thesis[7]. The information in this section is drawn from that document as well as a meeting at Landsvirkjun, that the author of this thesis attended, where the placement of a new well ÞG-15 at Þeistareykir was discussed.
Before a well can be placed a significant amount of information gathering has to take place and this data has to be organized and represented in a comprehensive way. The checklist, that Landsvirkjun is working on, requires the following data be available before a well can be placed

- A geographical map of the area including elevation mapping and satellite images.
- A geological map of the area including surface manifestations, visible faults and other surface expressions, temperature measurements of soil and a mapping of up flow zones.
- Geophysical measurements including magnetotelluric (MT) and transient electromagnetic (TEM) measurements, a gravity map of the area and a size estimation of the geothermal area.
- Geochemical analysis of fluid from up flow zones, estimate of the temperature of the system using geothermometers and the location of the main up-flow zones estimated based on this data.
- A geothermal potential estimate using Monte Carlo simulations or other similar methods.
- Data from wells that already have been drilled including core samples, down hole condition measurements, well imaging, temperature gradient and more.
- A conceptual model of the area including locations of major faults and location of permeable areas, temperature and pressure conditions as well as expected rock types and their location.

- Logistics including access to water for drilling, road access, well pad placements and more
- All necessary legal documents and permits including a land-use plan that specifies where Landsvirkjun is allowed to drill.
- Environmental assessment of drilling in the area.

Once all of these data have been gathered and the most important aspects of the data, for example the location of major faults and well paths, have been visualized a meeting is held to discuss the placement of the well. This meeting is attended by experts in their field that are familiar with the geothermal system in question. Their main focus will be to find a suitable well path for the well in order to allow for the most economical extraction of geothermal heat. A secondary objective is for the well to provide more information about the properties of the geothermal system and its extent. The experts will generally have an idea of where the up-flow zones, major faults and the main heat source are and will try to design a well path that intersects faults at a location where the geofluid will be suitable for extraction. Some well paths may be chosen that do not intersect these faults, or even be located at the edges of the geothermal system (wildcat wells), in order to gather further information although this is generally not done for the first few wells as the risk of failure is higher for this type of well.

1.2 Background and scope

This project was done as part of the research conducted by the Operations Research and Subsurface Modeling(ORSM) group which is a collaboration of Reykjavík University, the University of Iceland and Landsvirkjun. The inspiration for the project was the work of a former student Basil Jefferies[8]. Jefferies used the TOUGH2[9] reservoir simulator to simulate all possible wells in the Þeistareykir Geothermal field in the North of Iceland to determine the optimal well location. Jefferies' research was however specific to that one geothermal field and the aim of this project is to broaden the scope of that by seeing if it is possible to write an algorithm that simulates all possible wells in a geothermal field automatically to determine the optimal well location. This algorithm (hereby referred to as the Algorithm) would then be tested on reservoir models to determine the viability of using the Algorithm as a tool for decision makers when choosing well locations.

1.3 Literature review

Little effort has gone into developing tools that systematically find the optimal well placement in geothermal fields utilizing numerical forward modeling of production. Some of the research that has gone into related fields were reviewed and are briefly described here. Chen et al. (2014)[10] used a multivariate adaptive regression spline (MARS) technique to find the optimal design and placement for a well in Southern California, USA. Akin (2012)[11] used an inference method, artificial neural networks and a search algorithm to find the optimal placement of an injection well and tested it on the Kizildere Geothermal field in Turkey. The approach had some limitations as the results from the neural network did not corrolate well enough with the reference simulation used for validation. Jefferies (2016)[8] wrote an MSc thesis where the optimal well was chosen based on the evaluated net present values (NPV) for all possible well placements in a geothermal field.

The well placement problem is quite similar in the oil and gas industry as the results depend on a combination of geological, physical and economical parameters that have a complex relationship and are hard to grasp intuitively. The oil and gas industry has utilized algorithms quite extensively for well placement and a few of these will be briefly described here. Pan and Horne (1998)[12] investigated two multivariate interpolation algorithms, Least Squares and Kriging, to generate new realizations from a limited number of simulations in order to predict the optimal strategies in field development. Tupac et al. (2007)[13] presented a hybrid intelligent system to optimize oil field development. The system utilized evolutionary algorithms to optimize the positioning and characteristics of wells in a reservoir; distributed processing to perform simultaneous reservoir simulations; function approximation models as simulator proxies; and quality maps to use some reservoir information to improve the optimization process. Ding (2008)[14] presented a covariance matrix adaptation-evolution strategy(CMAES) method applied to the problem of well placement optimization. Onwunalu and Durlofsky (2009)[15] applied a particle swarm optimization (PSO) algorithm to determine the optimal well type and location. Minton (2012)[16] compares these methods and other common methods more extensively.

1.4 How can reservoir simulators be used to assist with well placement?

Forward modeling of production is currently not used widely for the decision making process of placing wells. Their primary role is to simulate the response the geothermal system has to utilization and how best to use the wells that are already in place or whose location has already been decided. Through the utilization of reservoir simulators and the Algorithm, decision makers are given quantitative data to rely on and consider on top of the qualitative data they will already have. This data can be used to suggest overlooked possibilities and give an estimate of the viability of a well being considered.

Chapter 2

Methods

The Algorithm uses the TOUGH2[9] numerical reservoir simulator along with Python[17] and PyTOUGH[18] to simulate possible well locations and evaluate their NPV in order to determine what the most optimal wells are. Once constructed the Algorithm was tested on a hypothetical model in order to determine the reliability of its output. A sensitivity analysis was performed using the hypothetical model, the processing time of the algorithm was also measured and compared using versions of the hypothetical model with a varying number of cells. This chapter goes through the methodology of constructing the Algorithm, what tools were used, what the structure of the Algorithm is, what assumptions are made and how testing was conducted.

2.1 Tools used

The first step in writing the Algorithm was to determine what tools were available and which ones to use. For this project TOUGH2 was chosen as a reservoir simulator, Python as a programing language and PyTOUGH utilized as an interface between the two. PetraSim was used for model construction and visualization.

2.1.1 TOUGH2

TOUGH2 is a numerical reservoir simulator designed primarily for geothermal systems but is also used for nuclear waste disposal as well as other applications. "[TOUGH2 simulates] nonisothermal flows of multicomponent, multiphase fluids in one, two and three-dimensional porous and fractured media" [9]. It does this by solving mass and energy balance equations for the mass and heat flows in the model. The thermophysical properties used in these equations are supplied by the appropriate equation of state (EOS) module during the simulation. For the purposes of this project EOS1 (water, water with tracer) was used as the majority of the fluid in hydrothermal systems will be water. By choosing EOS1 the model assumes that there are no dissolved solids in the system. TOUGH2 was originally developed in 1991 by the Earth Science Division of the University of California at Lawrence Berkley National Laboratory and is widely used in both industry and academia most often using a third party interface like PetraSim, TOUGH2Matlab or PyTOUGH due to the lack of an intuitive interface in the simulator itself.

2.1.2 Python 2.7

Python is a general purpose programing language developed by Guido van Rossum in the early 1990s. This language was chosen because it is a well-established programing language that is widely used, free of charge and has a wide variety of useful resources that can be added to it, including the PyTOUGH extension. Version 2.7 is chosen as it is the newest version that can utilize PyTOUGH.[18]

2.1.3 PyTOUGH

PyTOUGH is a Python extension designed as an interface for TOUGH2, it introduces routines that allow the user to create and manipulate TOUGH2 input files, process output files and call on TOUGH2 executables to run simulations. The extension was written by Dr. Adrian Croucher at the Department of Engineering Science at the University of Auckland, New Zealand. PyTOUGH can also be used to handle data from AUTOUGH2 which is the University of Auckland version of TOUGH2 [18]. This extension was chosen as it has functions and routines that add and extract data from TOUGH2 files that would otherwise have to be written into the Algorithm. The extension requires any 2.x version of Python that is newer than Python 2.4.

2.1.4 PetraSim

"PetraSim is an interactive pre-processor and post-processor for the TOUGH family of codes" [19]. This program is widely used in industry and has an interface that visualizes numerical models as they are changed making the program more intuitive than many other interfaces. This program was primarily used as a means to visualize models as well as create TOUGH2 input files for the models to input into the Algorithm. Additionally, the models used for testing, supplied by members of the ORSM group, were constructed using PetraSim and supplied as PetraSim files.

2.2 Algorithm structure

The Algorithm reads a numerical reservoir model, that the user supplies, and then proceeds to generate all the needed input files to simulate the possible wells. A possible well in this case is a cell in the model, the Algorithm generates one input file for every cell in the model each one with a single well added to a single cell. Once these files have been created the Algorithm proceeds to simulate all of them creating the output files that are then used to evaluate the net present value (NPV) of each possible well and arrange them into a list in order of NPV with the optimal well, defined as the well with the highest estimated NPV, at the top.
The Algorithm is split up into three main parts (PreProcessor, Processor and PostProcessor) as well as an algorithm (Generator) that generates scripts with the appropriate values for each run as well as making sure that the algorithms run in the correct order. The Algorithm uses base algorithms for the PreProcessor, Processor and PostProcessor the Generator opens these scripts, creates copies of them and adds the relevant variables to each one. A flow diagram that shows the order of operations can be seen on Figure 2.1. Generator is run first (and should in most cases be the only file that needs to be run)

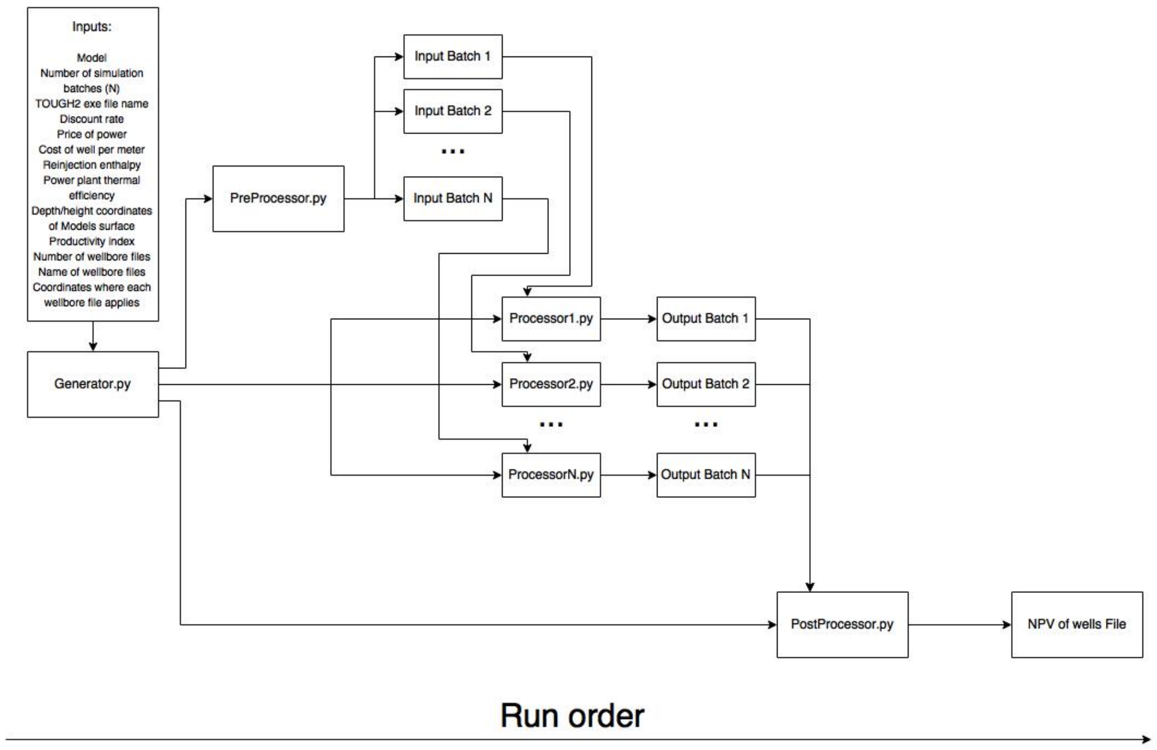


Figure 2.1: Flowchart describing how the Algorithm runs. The further to the right an operation is the later it is run. The Inputs are supplied by the user and each is described in Table 1. The user should run Generator.py which generates a PreProcessor.py script, a user defined number (N) of ProcessorXX.py scripts, a PostProcessor.py script and a batch file. The batch file makes sure the scripts are run in the correct order, first the PreProcessor.py, followed by all the ProcessorXX.py scripts and the PostProcessor.py script. PreProcessor.py creates the Input files needed for the simulations, these are sorted into N Input batches that are used as inputs for each of the ProcessorXX.py scripts. The Processor.py scripts each go into a simulation batch and run every Input file there creating N Output batches that contain the well data needed for the final step. The PostProcessor.py goes through the output files, evaluating the NPV of each well locations, and returns a file that lists all the simulated wells in order of their NPV, displaying their NPV, name and location coordinates within the model.

2.2.1 Inputs

Table 2.1 explains each of the inputs and their format.

Input Name	Format [units]	Format Example	Description
Model	"name of model".dat	Model.dat	A TOUGH2 model file for the geothermal system that is being simulated. This file needs to fulfill all the requirements of any other TOUGH2 model file and have geometric data for each element within the model.

Number of simulation batches	A positive integer (N)	4	This variable tells the algorithm how many simulations it should run simultaneously. This number should generally be equal to the number of processor cores that the computer running the Algorithm has. If the computer will be used for other activities while simulating this number should be 1 or 2 lower than the number of processor cores.
TOUGH2 exe file name	“name of executable”.exe	t2s_1.exe	This tells the Algorithm what the name of the TOUGH2 executable file that you want to use is. This will determine the EOS that the user wishes to use.
Discount rate	A number between 0 and 1	0.07	The assumed annual discount rate of the revenue stream from this single well. This needs to be a number between 0 and 1, not a percentage number.
Price of power	A number [$/MWh]	30	The assumed price the operator gets for their end product in $/MWh. This number can be in any currency but needs to be in the same currency as the number for “Cost of well per meter” and has to be the price the operator gets in that currency for every MWh.
Cost of well per meter	A number [$/m]	2000	The assumed cost of the well construction per meter of depth in $/m. This number can be in any currency but needs to be in the same currency as the “Price of power” and has to be the cost of the well in that currency for every meter of depth.
Reinjection enthalpy	A number [kJ/kg]	1000	The assumed enthalpy of the rejected geofluid. This would be the enthalpy of the fluid that is then reinjected or discarded in kJ/kg.
Power plant thermal efficiency	A number between 0 and 1	0.1	The assumed thermal efficiency of the power plant that receives the geofluid from the well. This needs to be a number between 0 and 1, not a percentage number. If the plant is using the fluid directly and not producing electricity, then this number can be set to 1.

Surface depth coordinates	A number [m]	2500	This number should represent the height or depth coordinates of the surface of the TOUGH2 model. Some models for example have the surface set at depth/height 0m while others may have the bottom depth set as 0m and therefore the surface at higher coordinates.
Productivity index	A positive number, usually of the format xEy for $x * 10^y$ [m^3]	2e-12	The assumed productivity index for the wells.
Number of wellbore files	A positive integer or 0	2	The number of wellbore files used for the simulations if any are available. If this is set to 0 the Algorithm will use the "well on deliverability" feature in TOUGH2.
Name of wellbore files	4 letter/number filename starting with f	F12b	TOUGH2 requires all wellbore file names to start with an f and PyTOUGH require these file names to four letters and/or numbers. These files are generated with wellbore simulators such as WellSim.
Coordinates where wellbore file applies	Array of 6 values defining the width, length and depth where the file applies [x1, x2, y1, y2, z1, z2]	100, 2500, 50, 700, 500, 1500	X, y and depth coordinates that each wellbore file applies to. The direction of these coordinates is determined by the setup of the model.

Table 2.1: Description of the Algorithms inputs.

2.2.2 Well types

The Algorithm uses two different types of production wells for its simulations, these are wells on deliverability and wells using wellbore files. The conditions where each type is used is discussed in section 2.2.6.

A well on deliverability uses an assigned bottom hole pressure in combination with the productivity index to calculate the flow in a well. A limitation of this is that the assigned bottom hole pressure will always be the same through the entire simulation. A second limitation is that PyTOUGH does not properly support this type of well and the bottom hole pressure cannot be assigned using the package, as a result the bottom hole pressure is set to 0 if a deliverability well is used. This low pressure will result in excessive flow rates in the wells

and will skew the results in favor of higher pressure cells as the pressure difference between the bottom hole pressure and the cell pressure will be greatest there. It was decided to keep this functionality in the algorithm as it will still give a relatively good idea of where the best wells would be located even if the NPVs will be unreasonably high.
A well on wellbore file uses an external file that defines the bottom hole pressure based on enthalpy and flow rate. These files are used to solve for flow rate and bottom hole pressure, based on the wells enthalpy, by iteration. This allows the bottom hole pressure to change with time resulting in a more realistic simulation. These files however have to be generated by a separate program, these programs are called well simulators. A well simulator simulates the flow of a well for a variety of enthalpies and flow rates thereby calculating the bottom hole pressure. A limitation of this is that these files only have a limited range and if the well achieves conditions outside this range the simulation will simply stop. The wellbore file used in testing the Algorithm is discussed in section 2.3.2.

2.2.3 Assumptions

In order to enable the Algorithm to run and estimate the NPV of the simulated wells certain assumptions had to be made concerning some of the parameters used. Many of the assumptions are made in order to simplify the subsequent calculations.
It is assumed that the TOUGH2 model is an accurate representation of the geothermal system as the models themselves do not show any uncertainty that can be used to show the confidence of the model or the variability from measured data. The price of power the operator gets is assumed to be known and that it is unchanged during the operation time. The cost of well construction is assumed to be known and that it is only dependent on the depth of the well, not being subject to the hardness of the rock, possible drilling and well construction complications. It is assumed that there are no operation and maintenance (O&M) costs associated with the well. Reinjection enthalpy is assumed to be the same for all possible well locations, this assumption is likely to skew the results as lower enthalpy wells would generally be able to reject a lower enthalpy than the higher enthalpy wells would.

2.2.4 Generator

This algorithm is used in order to generate the other parts of the Algorithm that then each run a specific part of the whole process. It takes the following inputs: model, number of simulation batches, TOUGH2 exe file name, discount rate, price of power, cost of well per meter, reinjection enthalpy, power plant thermal efficiency and the height/depth of surface, the remaining inputs are input into the PreProcessor.
Once the Generator has all the required inputs it creates a batch file called batch.bat and writes commands that run the PreProcessor, Processors and PostProcessor into it. The batch file is there only to run the algorithms in the correct order automatically, it will include one line for the PreProcessor, one line for each of the Processors and one line for the PostProcessor. The Generator then proceeds to open the base PreProcessor and make a copy of it called PreProcessor01.py, the base file has a placeholder that the Generator replaces with the following inputs in the copy: model and number of simulation batches. Similarly, the Generator will make a number of copies, equal to the number of simulation batches, of the base Processor, these files will be called Processor01py, Processor02.py and so on. These files will also have a placeholder that is replaced with the relevant inputs, in this case the following: model, number of simulation batches and TOUGH2 exe file name, the processor number (1 for Processor01.py, 2 for Processor02.py and so on) will also be added as an input here alt-

hough it is not user defined. The Generator will then create a copy of the base PostProcessor and create a copy of it called PostProcessor01.py, this file will include a placeholder that is replaced by the following inputs: model, number of simulation batches, discount rate, price of power, cost of well per meter, reinjection enthalpy and power plant thermal efficiency. Finally, the Generator runs the batch.bat file thereby starting the generation and processing of data.

2.2.5 Batch file

The batch file was written in order to allow for parallel processing of multiple TOUGH2 simulations, it makes sure that the scripts in the Algorithm run in the correct order and that the scripts that can be run in parallel run simultaneously. It starts by running the PreProcessor and waits until it has finished before proceeding to run all the Processors at the same time as well as the PostProcessor. While the PostProcessor requires output files from the Processors it is run at the same time as it includes a delay function that is discussed in Section 2.2.8.

2.2.6 PreProcessor

The PreProcessor generates the TOUGH2 input files that will be simulated. It opens the model that is input into the Algorithm, adds a generator to the first cell in the model and saves that configuration as Input000001, it then removes the well and adds a generator to the second cell and saves that configuration as Input000002, it goes through all the cells in the model like this creating a file for each configuration.
Before the PreProcessor starts generating input files it asks the user to define the simulation length in years, the assumed productivity index, in m3, as well as a user specified number of wellbore files. If the number of wellbore files is set to 0 all wells will be set up as wells on deliverability. If the number of wellbore files is larger than 0 the script will go through a short loop where the user specifies the name of the wellbore files and where in the model each applies. The PreProcessor uses this information to setup a series of lists that it can call on later to determine what type of well should be used in any given area.
Next the PreProcessor defines the starting time as 0 and the end time is set to the user specified number of years that are to be simulated. some of the parameters the model that is to be simulated will have.
In order to keep the folder the Algorithm is running in (from here on referred to as the root folder) as tidy as possible the PreProcessor will create a folder called "temp" within the root folder. The PreProcessor will also create subfolders within the "temp" folder called "simulator1", "simulator2",... that will contain each of the simulation batches. The PreProcessor will move all the wellbore files that have been specified to each of the "simulator" folders.
The PreProcessor will now proceed to generate the input files. It does this, like mentioned above, by opening the model that will be simulated, adding a well to the first cell, saving the configuration as a copy, removing the well and moving on to the next cell. Each time the PreProcessor places a well it starts by finding the center of the cell it is adding the well to, if this cell is within an area defined as an area where a wellbore file applies the PreProcessor will name the well type in the appropriate way so that TOUGH2 will call on the correct wellbore file during simulation. All the generated input files are added into the "temp" folder.

2.2.7 Processor

Once the PreProcessor has generated all the input files the Processors start running. They will each begin with moving the input files assigned to them into the appropriate "simulator" folder, Processor01 will move the files assigned to it (deleting the original copy to avoid duplicates and save hard drive space) into the "simulator1" folder, Processor02 will move the files assigned to it into the "simulator2" folder and so on. The Processors will also copy the TOUGH2 executable file from the root folder to each of the "simulator" folders, this is done to allow for parallel processing as trying to run the same executable file for multiple simulations simultaneously will result in an error. Before starting its simulations, the Processors will also copy all wellbore files into each of the "simulator" folders.
Once all the relevant files have been copied to the correct folders the Processors will start simulating the input files assigned to them. This will then generate output files called Input000001.listing for the first simulation, Input000002.listing for the second simulation and so on.

2.2.8 PostProcessor

The PostProcessor will start running at the same time as the Processors will in order to save some time. It will start by going through each of the "simulator" folders and move all output files into the "temp" folder. If the PostProcessor attempts to move an output file that is still being generated it will run into an error and stop, to prevent this the PostProcessor starts by checking if the next file in sequence exists. By doing this it is ensured that the files being copied are not being used by the Processors and that a ready output file is what is copied. Once all the output files have been copied into the "temp" folder the PostProcessor will start reading them and evaluate the NPV of each well location. It starts with Input000001.listing then goes to Input000002.listing and so on.
The NPV of each well location is calculated using Equation 2.1 where n is the number of output times, t_i is the simulation time at which output i is printed in years, $Revenues_i$ are the revenues from time t_{i-1} to time t_i, $Costs_i$ are the costs from time t_{i-1} to time t_i and r is the discount rate.

$$NPV = \sum_{i=0}^{n} \frac{Revenues_i - Costs_i}{(1+r)^{t_i}} \tag{2.1}$$

For the sake of simplicity, the costs are assumed to be 0 for all years except the first one where the only costs are assumed to be the cost of constructing the well. The cost of the well is assumed, as mentioned above, to only be dependent on the depth of the well. To determine the depth of the well the Algorithm finds the coordinates of the center of the cell that has the well, that is being simulated in each case, and subtracts that from the user defined surface depth coordinates. This depth is then simply multiplied by the cost of well per meter to give the cost of the well which will be input as $Costs_0$ in Equation 2.1.
In order to make a conservative estimate of the NPV the revenues are always calculated using the well conditions at the end of time period. The revenues are calculated from the well flow and enthalpy using Equation 2.2 where $\dot{m}_i$ is the mass flow of geofluid in at time t_i, h_i is the enthalpy of the geofluid at time t_i, h_r is the enthalpy of rejected geofluid (or reinjected), η is the thermal efficiency of the power plant and Price is the price of power to the operator.

$$Revenues_i = (h_i - h_r) * \dot{m} * \eta * (t_i - t_{i-1}) * Price \tag{2.2}$$

As mentioned in 2.2.3 the assumption that the rejected enthalpy is the same for all wells will result in skewed results for some wells as lower enthalpy wells would be able to reject lower enthalpy fluids thereby extraction proportionally more energy than the Algorithm calculates. Further the revenues can result in a negative number if the well enthalpy is lower than the rejected enthalpy.
Once the NPV has been calculated for an individual well the PostProcessor will save the NPV as well as the name of the well, its location, the number of the simulation file and the last simulated output time to a list, the NPV for all wells are stored in the same list. When the NPV for all wells has been calculated the list is arranged in order of NPV with the highest NPV well at the top, this list is then output into a text file called "NPVs of wells".

2.3 Testing

In order to test the Algorithm a hypothetical model was used, this model was run through the Algorithm as well as being used in sensitivity analysis to determine what numerical parameters influenced the results the most. The following sections briefly describe the model that was used, the sensitivity analysis and the testing done to evaluate the effect cell number has on the processing time, the results of these analysis is shown in Chapter 3 Results. All testing of the Algorithm was done on the authors personal computer parallel processing four simulations at a time, the technical specifications of the computer are specified in Appendix B.

2.3.1 Hypothetical model

The hypothetical model (hereby referred to as the Hypothetical model) was originally made by Egill Júlíusson and then modified by the author. All work on this model was done in PetraSim.
This is a fairly simple model that includes three rock types, a base rock, a cap rock and a reservoir rock. The model geometry is cubical with a width of 10,000m in both x and y directions and a depth of 2,500m with 3050 cells. The thermal and physical properties of each rock type can be seen in Table 2.2, only the permeability of the rock types varies in this model. A picture of the model showing the various rock types as well as the location of two corner points and their coordinates can be seen on Figure 2.2

Rock type	Density $[kg/m3]$	Porosity $[-]$	XY-Permeability $[m^2]$	Z-Permeability $[m^2]$	Wet heat conductivity $[W/m*K]$	Specific heat $[J/kg*K]$
Cap rock	2600.0	0.1	1.0*10-15	1.0*10-15	2.0	1000.0
Reservoir rock	2600.0	0.1	1.0*10-14	1.0*10-14	2.0	1000.0
Base rock	2600.0	0.1	1.0*10-17	1.0*10-17	2.0	1000.0

Table 2.2: Properties of the three rock types in the Hypothetical reservoir model used for testing the Algorithm.

Both the side and bottom boundaries are given fixed conditions. The side boundaries of the model are set to fixed conditions of pressure and temperature which are described by a

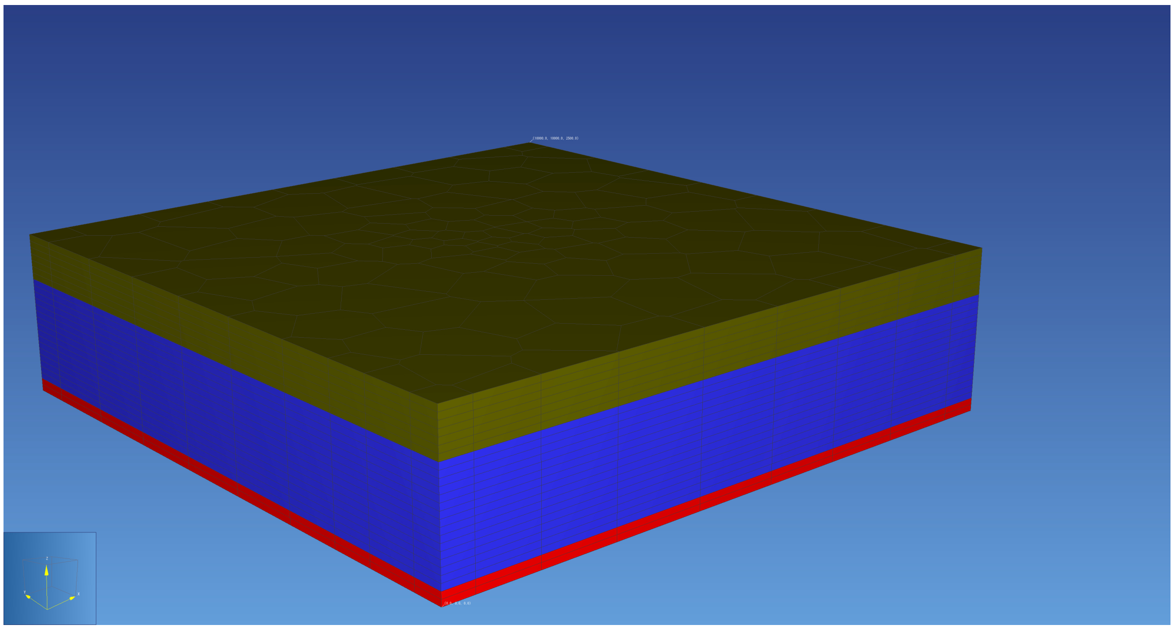


Figure 2.2: Visual representation of the Hypothetical model, used for testing the Algorithms, showing the three different rock types and their locations within the model. The top layers are the cap rock represented with a brown color, the reservoir rock is represented with a blue color and finally the bottom layers consist of the base rock represented with a red color. Two corner points are shown on the figure as well as their coordinates, these are at the bottom the point (0.0; 0.0; 0.0) and at the top the point (10,000.0; 10,000.0; 10,000.0)

temperature gradient and a pressure gradient (linear equations that are only dependent on depth). As origin of the model is at its bottom the depth is defined as a height from bottom as opposed to the more standard way of defining it as a depth from surface. The temperature gradient for the boundary can be seen on Equation 2.3, it sets the temperature at the surface to 10°C, the temperature at the bottom to 100°C with a first degree linear relation between the two where z is the height from the bottom in meters. This is the equivalent of having 10°C temperature on the surface and a temperature gradient of 36°C/km.

$$T(z) = 100°C - 0.036z°C/m \tag{2.3}$$

The pressure gradient is set so that the pressure at the surface is equal to 100kPa, which is close to the average atmospheric pressure, and increases by 7.4kPa per meter, which is about 25% below the hydrostatic pressure gradient. The equation that describes the pressure can be seen in Equation 2.4

$$P(z) = 1.86 * 10^7 Pa - 7400z Pa/m \tag{2.4}$$

The center cells in the bottom layer also had fixed boundary conditions. These cells were all cells with a center between x=3333m and x=6666m and between y=3333m and y=6666m. These cells had the same pressure as the ones at the outer boundary, as defined above, but had a fixed temperature of 450°C. These cells then act as a heat source for the system.
Once all of these conditions have been set the model can be run through PetraSim in order to achieve steady state conditions. Figure 2.3 shows plane slices of the temperature in the model at steady state.

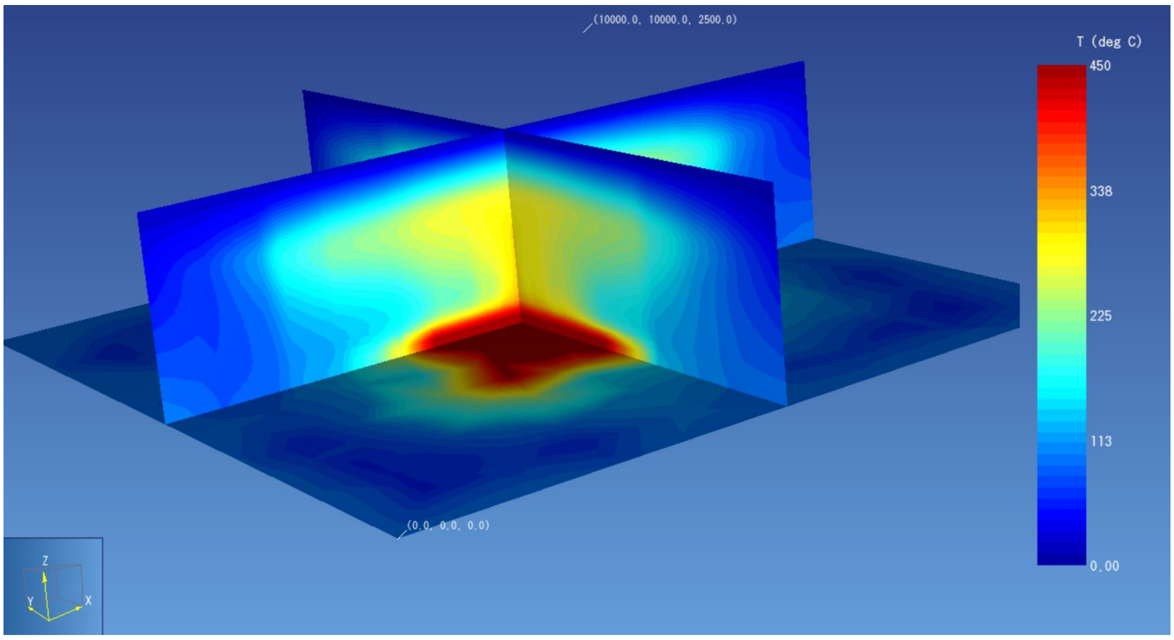


Figure 2.3: Plane slices showing the temperature of the Hypothetical model at steady state.

2.3.2 Running the Algorithm

The Hypothetical model was run through the Algorithm with all wells set to well on deliverability on the one hand and with a wellbore file on the other. The wellbore file that was used is from well BJ-14 at Bjarnarflag in northern Iceland. This file was lent to the author by ISOR (Iceland GeoSurvey) for research purposes along with several other files including the wellbore files for BJ-11 and BJ-13. The file for BJ-14 was chosen for use on the basis that it had the largest enthalpy range of the available files. This file was made by Sigríður Sif Gylfadóttir in 2013 using wellbore simulation to mimic the output from the well, Figure 2.4 shows the simulated well bottom pressure of the well as a function of enthalpy and mass flow.

The Algorithm was run using the same values for numerical inputs in both the well on deliverability and the wellbore file case. The values used were the following

- The productivity index of 2.5*10-12 m^3 was chosen as it is in within the simulated and calculated ranges at Bjarnarflag.[20]

- Discount rate is set to 0.07 (7%) as this is a discount rate that Landsvirkjun would most probably use.[21].

- The price of power to the operator is set to $30/MWh, this value is based on the average LCOE of new geothermal plants as estimated by the EIA[3]. The value estimated by the EIA is $45.0/MWh as that accounts for all parts of the power plant this number needs to be lowered as the algorithm only accounts for the well itself. To do this the cost as estimated by the EIA was halved and rounded to the nearest whole $5 as this is a very rough estimate.

- The cost of the well is set to $2,000/m and is based on estimates from Kipsang (2013)[22]. Kipsang used a 3,000m sample well and estimated the cost of that well to be $6,045,000 or roughly $2,000/m.

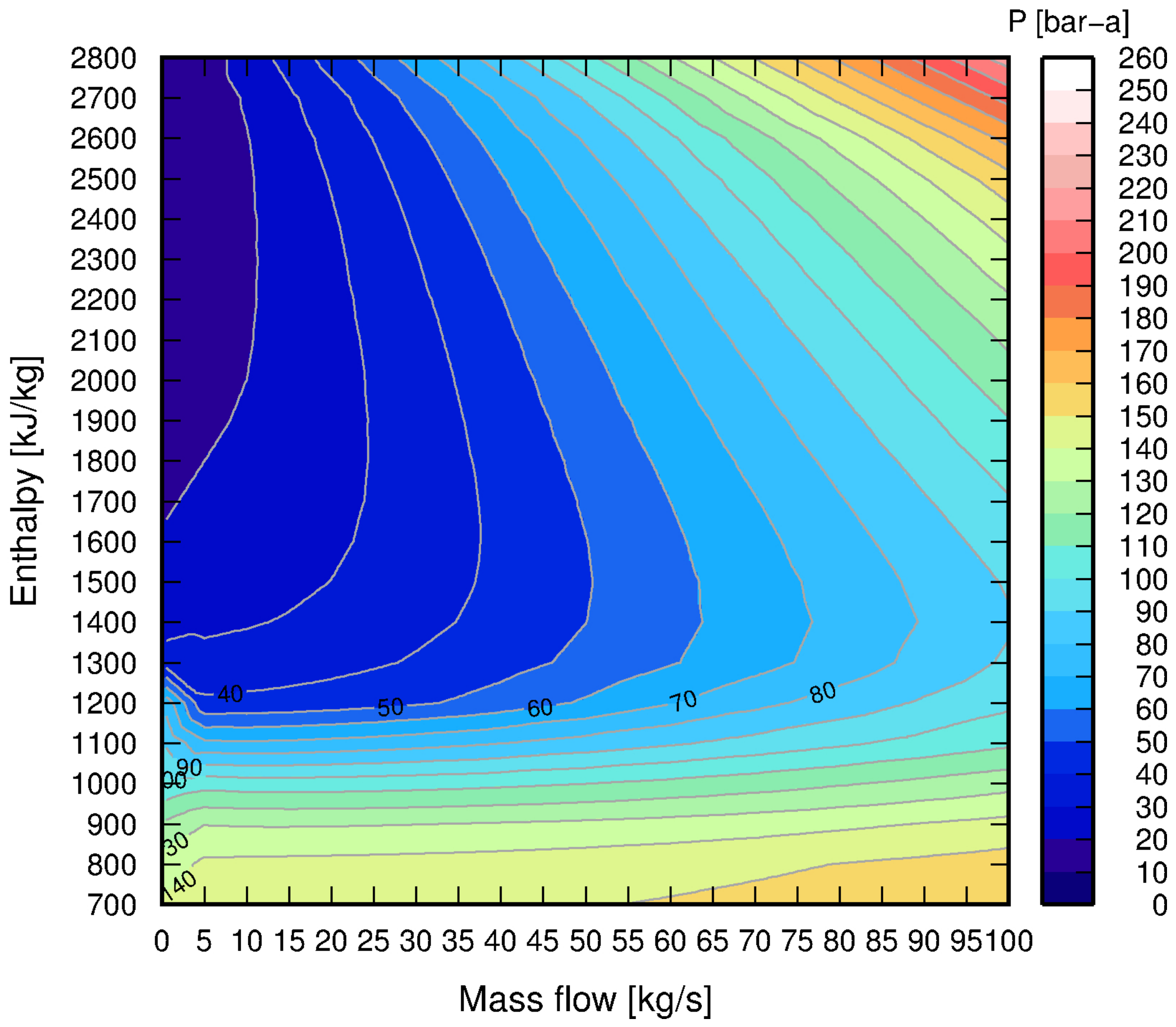


Figure 2.4: Well bottom pressure as a function of mass flow and enthalpy for well BJ14 in the Bjarnarflag geothermal field[20].

- The rejected enthalpy is set to 1000kJ/kg. This value is chosen as a compromise between getting unreasonably high NPVs by setting the value too low and getting unreasonably few or no positive NPVs by setting it too high.

- The thermal efficiency is set to 0.1 (10%) and is chosen as a lower end efficiency based on results from Moon & Zarrouk (2012)[23].

2.3.3 Sensetivity analysis

In order to evaluate the effect each of the Algorithms numerical inputs has on the end results a sensitivity analysis was run. Each of the Algorithms numerical inputs were increased individually by 10% and run through the Algorithm, these values are productivity index, discount rate, price of power, cost of well, rejected enthalpy and thermal efficiency of power plant. The outputs were evaluated with two parameters, the highest NPV out of all simulated wells and the number of positive NPVs out of all simulated wells. The analysis was on one hand done with all wells set to well on deliverability and with a wellbore file on the other.

2.3.4 Processing time

In order to give an idea of the processing time required for the algorithm several runs using the Hypothetical model with varying number of cells were done. Each version of the model was run two times, once for the well on deliverability setup and once for the wellbore file setup. The number of cells used for this analysis was 1350, 1550, 2200, 3050, 5050 and 6700.

Chapter 3

Results

This chapter presents the results from running the Hypothetical model with the Algorithm as well as the sensitivity analysis and processing time analysis.

3.1 Optimal well placement

The optimal well placement as discussed in Chapter 2 is the well with the highest NPV out of all simulated wells. This chapter will discuss the top wells and their locations in the well on deliverability case and the wellbore file case.

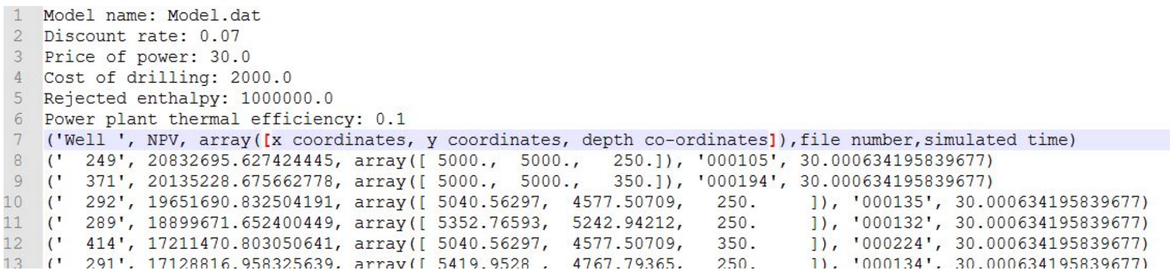
3.1.1 Well on deliverability

For this case the Algorithm estimated 255 potential wells with a positive NPV, the associated cell name, NPV and coordinates of the top five is shown in Table 3.1, part of the output from the Algorithm can be seen on Figure 3.1

Cell name	NPV	X-coordinates	Y-coordinates	Depth from surface
249	$20,832,695.63	5000m	5000m	2250m
371	$20,135,228.68	5000m	5000m	2150m
292	$19,651,690.83	5040.56297m	4577.50709m	2250m
289	$18,899,671.65	5352.76593m	5242.94212	2250m
414	$17,211,470.8	5040.56297m	4577.50709m	2150m

Table 3.1: Cell name, NPV, XY-coordinates and the depth from surface for the top five best wells in the Hypothetical model using the well on deliverability setup.

As can be seen from the X- and Y-coordinates all of these cells line up in the center of the model with varying depth. The well in cell 249 had the highest NPV of $20,832,695.63, it is at a significant depth within the system and is in the lowest reservoir rock layer, just above the base rock layer. As the base rock layers have a defined fixed state the NPV of this well may be somewhat misleading due to the proximity to an endless source of high temperature and pressure, this also applies to 292 and 289 as they are in the same layer and above the fixed state heat source.
The top five best wells are all located in the bottom two layers of the reservoir, right above the base rock, and around the center of the system, right above the heat source. As was mentioned in Chapter 2 these wells will have an unreasonably high NPV as their bottom



```
Model name: Model.dat
Discount rate: 0.07
Price of power: 30.0
Cost of drilling: 2000.0
Rejected enthalpy: 1000000.0
Power plant thermal efficiency: 0.1
('Well ', NPV, array([x coordinates, y coordinates, depth co-ordinates]),file number,simulated time)
('  249', 20832695.627424445, array([ 5000.,  5000.,   250.]), '000105', 30.000634195839677)
('  371', 20135228.675662778, array([ 5000.,  5000.,   350.]), '000194', 30.000634195839677)
('  292', 19651690.832504191, array([ 5040.56297,  4577.50709,   250.     ]), '000135', 30.000634195839677)
('  289', 18899671.652400449, array([ 5352.76593,  5242.94212,   250.     ]), '000132', 30.000634195839677)
('  414', 17211470.803050641, array([ 5040.56297,  4577.50709,   350.     ]), '000224', 30.000634195839677)
('  291', 17128816.958325639, array([ 5419.9528 ,  4767.79365,   250.     ]), '000134', 30.000634195839677)
```

Figure 3.1: First 12 lines of the output file from the Algorithm in the well on deliverability case.

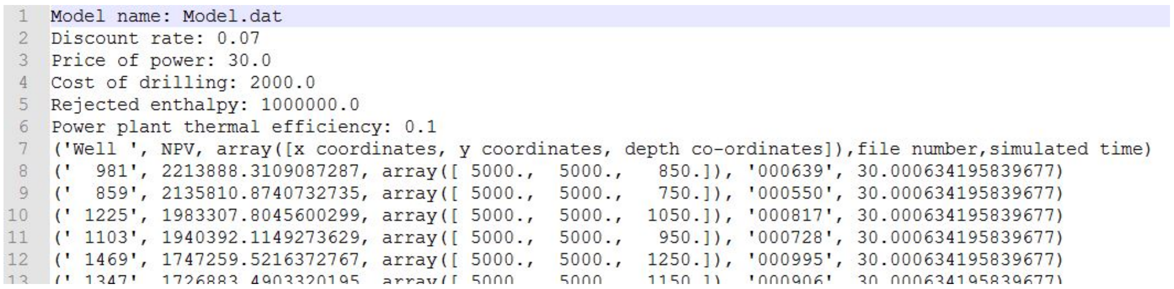
hole pressure is set to 0, they do, however, give a good idea of the location of the optimal wells. These locations (at depth above the heat source) are not surprising and would most probably be some of the first well locations considered as targets for drilling in a geothermal system like this.

3.1.2 Well on wellbore file

For this case the Algorithm estimated 24 potential wells with a positive NPV, the associated cell name, NPV and coordinates of the top five is shown in Table 3.2, part of the output from the Algorithm can be seen on Figure 3.2

Cell name	NPV	X-coordinates	Y-coordinates	Depth from surface
981	\$2,213,888.311	5000m	5000m	1650m
859	\$2,135,810.874	5000m	5000m	1750m
1225	\$1,983,307.805	5000m	5000m	1450m
1103	\$1,940,392.115	5000m	5000m	1550m
1469	\$1,747,259.522	5000m	5000m	1250m

Table 3.2: Cell name, NPV, XY-coordinates and the depth from surface for the top five best wells in the Hypothetical model using the well on wellbore file setup.



```
Model name: Model.dat
Discount rate: 0.07
Price of power: 30.0
Cost of drilling: 2000.0
Rejected enthalpy: 1000000.0
Power plant thermal efficiency: 0.1
('Well ', NPV, array([x coordinates, y coordinates, depth co-ordinates]),file number,simulated time)
('  981', 2213888.3109087287, array([ 5000.,  5000.,   850.]), '000639', 30.000634195839677)
('  859', 2135810.8740732735, array([ 5000.,  5000.,   750.]), '000550', 30.000634195839677)
(' 1225', 1983307.8045600299, array([ 5000.,  5000.,  1050.]), '000817', 30.000634195839677)
(' 1103', 1940392.1149273629, array([ 5000.,  5000.,   950.]), '000728', 30.000634195839677)
(' 1469', 1747259.5216372767, array([ 5000.,  5000.,  1250.]), '000995', 30.000634195839677)
(' 1347', 1726883.4903320195, array([ 5000.,  5000.,  1150.]), '000906', 30.000634195839677)
```

Figure 3.2: First 12 lines of the output file from the Algorithm in the well on wellbore file case.

As can be seen from the X- and Y-coordinates all of these cells line up in the center of the model however in this case they are not at the same depth as in the deliverability case. The well in cell 981 had the highest NPV of \$2,213,888.311 and is located in the center of the model, above the heat source, at a depth of 1650m. This location, as well as the other

locations in the top five, is within the lower part of the upper heat zone seen on Figure 2.2 as the yellow area in the middle of the figure. These locations are to be expected as some of the best locations as they are within a high temperature zone in the middle of the system and would most likely be some of the first locations considered for drilling. Unlike the top five wells in the deliverability case these wells form a line along the middle of the system as opposed to a plane around a certain depth. This is at least in part due to the fact that the bottom hole pressure in the wellbore case will be different for each well thereby not skewing the results in favor of higher pressure/deeper wells.
None of the top wells are located in the bottom layers closest to the heat source in this case. This is because of the conditions in those cells exceeding the wellbore file data set resulting in the simulation being terminated prematurely. Only 243 simulation simulated more than a year in this case and 1237 simulated more than 0 years. The 994 simulations that simulated more than 0 years but less than 1 year all reached equilibrium within the first ten time steps and therefore terminated the simulation printing out the results at time step ten. These cells are all cells with fixed conditions and it is therefore normal that no change takes place there.

3.2 Sensetivity analysis

For the sensitivity analysis each of the numerical inputs were increased by 10% and the change in NPVs was measured. The values that were used to represent the change in NPVs were the value of the highest NPV and the number of positive NPVs for each case.

3.2.1 Well on deliverability

The results from the sensitivity analysis of the well on deliverability case can be seen on Figure 3.3.

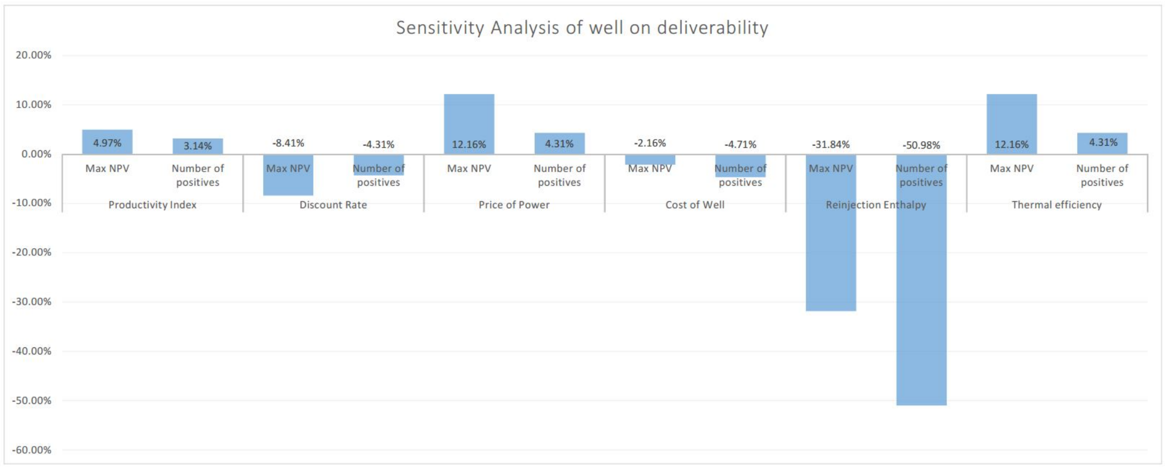


Figure 3.3: Results of sensitivity analysis of the well on deliverability case. Each input was increased by 10% and the change in the highest NPV as well as the number of positive NPVs was measured.

As Figure 3.3 shows the results are more sensitive to some inputs than others. The reinjection enthalpy is by far the most sensitive parameter with a 31.84% reduction in the highest NPV and a 50.98% reduction in the number of positive NPVs. The impact of this parameter is very dependent on the enthalpy of the fluid coming out of the well, if the difference between the reinjection enthalpy and well enthalpy is small then the impact will

be larger while a larger difference will lessen the impact. In this case the enthalpy of the well with the highest NPV, in cell 249, has an enthalpy of 1,661 kJ/kg at time step 1. This means that the reinjection enthalpy of 1,000 kJ/kg is fairly close to the well enthalpy resulting in the large impact of this parameter.
The second and third most sensitive parameters are the price of power and thermal efficiency of the power plant, both increasing the max NPV by 12.16% and the number of positives by 4.31%. This is to be expected as both values will affect the revenue streams in the same way, that is the price of power will increase the revenue by increasing the base price of power per MWh while the thermal efficiency will increase the amount of power that can be sold. As these values are multiplied at the same stage in the calculations they will always have the same proportional impact on the final NPVs.
The discount rate is the fourth most sensitive parameter reducing the max NPV by 8.41% and the number of positives by 4.31%. Following that are the productivity index, with a 4.97% increase in max NPV and a 3.14% increase in the number of positives, and cost of well, a 2.16% decrease in max NPV and a 4.71% reduction in the number of positives. This is somewhat unexpected as the productivity index controls the flow from the wells and would be expected to have a more significant impact, this may however be lessened somewhat because of discounted value of the revenue streams over the long period. It is also probable that the unreasonably high flow in the wells, caused by the low bottom hole pressure, causes the productivity index to have less of an impact on the results. The cost of well was also expected to be a more sensitive parameter as it plays a major role in the NPV calculations and is the only source of costs used in the simulation. This seems to be outweighed by the other parameters that affect the positive revenue streams resulting in the lower impact of the cost of well.

3.2.2 Well on wellbore file

The results from the sensitivity analysis of the well on wellbore file case can be seen on Figure 3.4.

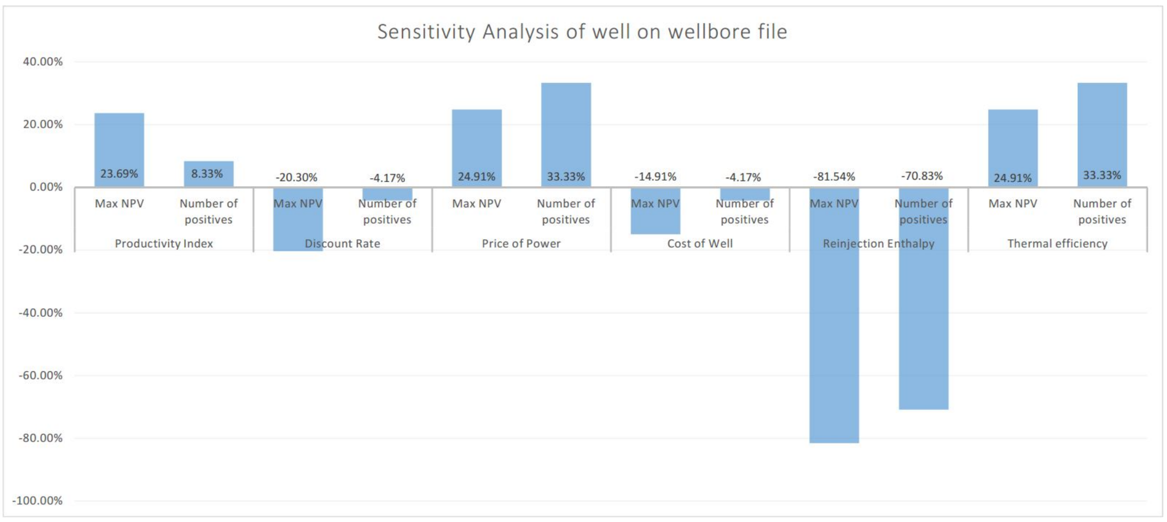


Figure 3.4: Results of sensitivity analysis of the well on deliverability case. Each input was increased by 10% and the change in the highest NPV as well as the number of positive NPVs was measured.

As Figure 3.4 shows the results are more sensitive to some inputs than others. The reinjection enthalpy is by far the most sensitive parameter with an 81.54% reduction in the highest NPV and a 70.83% reduction in the number of positive NPVs. Similar to what was discussed in Section 3.2.1 this is due to the small difference between the wells enthalpy and the reinjection enthalpy, it is however exaggerated here as the highest NPV well, in cell 981, only had an enthalpy of 1,371 kJ/kg.
The second and third sensitive parameters are the price of power and thermal efficiency in this case as in the well on deliverability case, this was already discussed in Section 3.2.1.
The fourth most sensitive parameter in this case is the productivity index, with a 23.69% increase in max NPV and an 8.33% increase in the number of positives, this is more in line with what one would expect than what was seen in the well on deliverability case. The fifth most sensitive parameter is the discount rate, with a 20.30% reduction in max NPV and a 4.17% reduction in the number of positives, the reduction in the number of positive NPVs is very similar to that in the well on deliverability case but the reduction in the highest NPV is significantly more. This is due to the smaller revenue streams being more affected by the discount rate than the larger ones in the well on deliverability case.
The least sensitive parameter is again the cost of well, with a 14.91% reduction in max NPV and a 4.17% reduction in the number of positives, this is significantly more than was seen in the well on deliverability case and can again be attributed to the smaller revenue streams, in the wellbore file case, resulting in the initial cost having a proportionally higher impact on the NPVs.

3.3 Processing time

As mentioned in Chapter 2 a processing time analysis was done on the Algorithm for models of different sizes. If the Algorithm can be run on a large detailed numerical model in a reasonable time frame it may become a valuable tool for decision makers in the future. By looking at the processing time for different model sizes a relationship between the number of cells in the model and the processing time can be found, this relationship however is only valid for this specific model and the specific computer it was run on, it will however give some indications as to how long the processing time might be. The results from the measurements were plotted on a log-log scale and fitted to a linear equation as that seemed to give the best fit, the plot along with the measured data as well as the fitted data and the equation for the fit are shown on Figure 3.5. The equations for the fitted functions are shown below, Equation 3.1 for the well on deliverability case and Equations 3.2 for the well on wellbore file case, where t represents the processing time in seconds and n represents the number of cells in model.

$$t = 10^{-5.6419} * n^{2.2622} \tag{3.1}$$

$$t = 10^{-5.6638} * n^{2.174} \tag{3.2}$$

As can be seen on Figure 3.5 are a close fit to the data with a calculated RMS error of 21.21 for the well on deliverability and 10.60 for the well on wellbore case. It should however be noted that in the case of the wellbore file most simulations were terminated before the end time due to the limited range of the wellbore file used, this results in somewhat lower processing times.
Using the equation for the well on deliverability (as that gave the higher time estimates) a

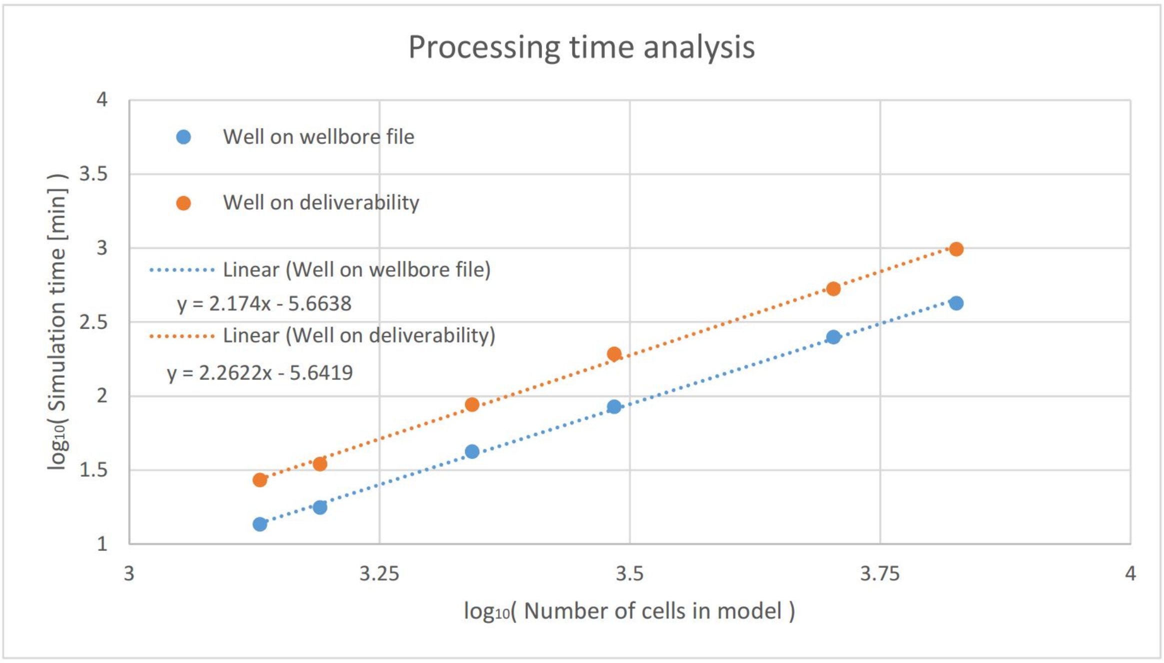


Figure 3.5: Results of the processing time analysis done using the Hypothetical model with the Algorithm. Results are shown for both the well on deliverability and well on wellbore file cases along with a fitted trendline and its equation. Note that the trendlines are fitted to the logarithmic data and not the original data and the x value has to be the logarithm of the number of cells and will give the logarithm of the processing time as a result.

similar model with 13,500 cells, which is close to the size of the Hágöngur model made by Ximena Guardia Muguruze (Muguruze, 2015), would take approximately 84 hours. However, it is also important to keep in mind that the Hypothetical model is a simple model and is therefore likely to be more stable and solve faster than most models of real geothermal systems. The Hágöngur model on the other hand has a more complicated geological structure and it is therefore likely that the processing time would be longer. Processing of data in geothermal fields sometimes take weeks or even months to do and if a numerical model of the system exists then even a few days of processing to get the additional data from the Algorithm may be feasible. The Algorithm is well suited for parallel computing and utilizing this technology, that is now readily available, the processing time could be shortened.

Chapter 4

Discussion

The Algorithm has been tested on a model and shown some reasonable results although some of the limitations are also apparent. These are discussed briefly in the following chapter along with a conclusion and suggestions for future work on the Algorithm.

4.1 Most important parameters

In the Hypothetical model that was used to test the Algorithm the results were most sensitive to the reinjection enthalpy. This parameter is hard to control and will largely be restricted by the geochemistry of the fluid. The second and third most sensitive parameters are the price of power and thermal efficiency of the power plant. These are somewhat easier to manage as they can be estimated with relative ease. The thermal efficiency can be influenced by the equipment that is used in the power plant, however, more efficient plants tend to have a higher capital cost and the cost and technical capabilities will be the largest restrictions on this parameter. The price of power is in many cases changing in time but this can be mitigated by having a pre-established power purchase agreement with a buyer.

4.2 Dependency on data reliability

Many assumptions have to be made in order to run the Algorithm and the results from it are only as good as the assumptions that are made. A limited knowledge of the geothermal system in question may result in a variety of unreliable estimations along the way. As the Algorithm utilizes a numerical model of the geothermal system the model needs to be reasonable in order for the Algorithm to give valuable results. One way to mitigate the error associated with the model itself would be to integrate this error into the Algorithm and calculate the risk associated with each well, this would then give investors and operators some idea of where the safest valuable wells are located. Similarly, if wellbore files are used they have to fit the system in question as results of the simulation can give results outside the wellbore files range and thereby not run to the end as was seen in the case of the well on wellbore file discussed in Section 3.1.2.

4.3 Summary

The Algorithm utilizes the Python programing language in combination with the PyTOUGH package in order to create and simulate a large ensemble of possible wells in a numerical

TOUGH2 model of a geothermal system. It uses these data to evaluate the NPV of all possibilities and return a list arranged in order of NPV. The algorithm was tested on a simple hypothetical model using the well on deliverability feature as well as a wellbore file from well BJ-14 in the Bjarnarflag geothermal system Iceland. The well on deliverability case successfully ran all simulations and gave the highest NPVs at depth in the center of the reservoir over the heat source. The well on wellbore file case successfully ran 243 simulations and gave the highest NPVs along the center or the model at a depth of 1250-1750m from surface, in the upper heat zone shown on Figure 2.2. The processing time of the simulations as a function of the number of cells in the model was also measured and found to fit to a second degree polynomial very closely. From the fitted data an estimated processing time of 84 hours for a similar model with 13,500 cells is found. This processing time may not be representative of a realistic model however due to the simplicity of the Hypothetical model. However, in comparison to the weeks or months that data processing in the geothermal industry can take this time frame of days seems a feasible option.

4.4 Conclusion

The Algorithm is functional despite several limitations and can create a list of optimal wells based on the supplied model. This process is (as are all geothermal reservoir models) subject to many assumptions and is highly sensitive to data reliability and availability, this is especially true for the reliability of the model itself. Further the Algorithm runs within a reasonable time frame and could therefore be a viable tool for decision makers during the well placement process.

4.5 Future work

While the Algorithm works it is far from perfect and a lot of work could go into improving it. An optimization of the Algorithm itself may be useful as well as further utilizing parallel processing for the pre- and post-processing steps. Restrictions on possible well locations to reduce the amount of simulations run would help with cutting down the processing time of the Algorithm, this could include things like a minimum and maximum depth of wells, disallowing wells in cells at fixed conditions or too close to boundaries. Implementing the error associated with each well to give an idea of the risk involved with each location would give valuable data to the decision makers both on what the safest well choices are and indications as to areas that may benefit from further exploration. Adding in more options for the user, for example by having the productivity index be defined for any given area separately similar to how the area a wellbore file applies to is defined. This should allow for more realistic estimation of the optimal wells and their NPV.
Running the Algorithm with a version of TOUGH2 that is coupled with a wellbore simulator would also be a big step forward. That way much of the problems with providing an appropriate wellbore file for the given depth and design of each well could be eliminated.
It is important to test the Algorithm with an accepted model of an actual geothermal system in order to further identify potential problems and evaluate feasibility. If the simulation time can be brought down significantly enough, or access to sufficient processing power is available, the Algorithm could potentially be modified to find the optimal combination of multiple wells. This could for example be done by picking the top few wells and running simulations with each of them along with all other possibilities in order to find the best combination.

This work could also benefit from adding criteria that identifies wells as close to identical so that simulations that would have two or more wells in close proximity to each other would be deemed unfit for simulation.

Bibliography

[1] T. Pratchett, *The wee free men*. 2003.

[2] REN21, "Renewables 2016 global status report", Renewable Energy Policy Network for the 21st Century, Tech. Rep., 2016.

[3] U. E. I. Administration, "Levelized cost and levelized avoided cost of new generation resources in the annual energy outlook 2016", U.S. Department of Energy, research rep., 2016.

[4] ——, "Updated capital cost estimates for utility scale electricity generation plants", U.S. Department of Energy, research rep., 2013.

[5] M. Gehringer and V. Loksha, "Geothermal handbook: Planning and financing power generation", Energy Sector Management Assistance Program, 2012.

[6] IFC, "Success of geothermal wells: A global study", International Finanace Corporation, research rep., 2013.

[7] Landsvirkjun, "Placement and drilling of a well in a geothermal system", 2016.

[8] B. A. I. Jefferies, "Optimal well placement in the theistareykir geothermal field for the next well in succession", Master's thesis, Reykjavik University, 2016.

[9] K. Pruess, C. Oldenburg, and G. Moridis, *Tough2 user's guide, version 2.0*, 1999.

[10] M. Chen, A. F. Tompson, R. J. Mellors, and O. Abdalla, "An efficient optimization of well placement and control for a geothermal prospect under geological uncertainty", *Applied Energy*, Oct. 29, 2014.

[11] S. Akin, M. V. Kok, and I. Uraz, "Optimization of well placement geothermal reservoirs using artificial intelligence", *Computers & Geosciences*, 2012.

[12] Y. Pan and R. Horne, "Improved methods for multivariate optimization of field development scheduling and well placement design", *Society of Petroleum Engineers*, 1998.

[13] Y. T. et al., "Evolutionary optimization of oil field development", *Digital Energy Conference and Exhibition*, 2007.

[14] Y. Ding, "Optimization of well placement using evolutionary methods", *Europec/EAGE Conference and Exhibition*, 2008.

[15] J. Onwunalu and L. Durlofsky, "Application of a particle swarm optimization algorithm for determining optimum well location and type", *Springer Science*, 2009.

[16] J. J. Minton, "A comparison of common methods for optimal well placement", University of Auckland, research rep., 2012.

[17] M. Lutz, *Learning python*. 2013.

[18] A. Croucher, *Pytough user's guide*, 2015.

[19] T. Engineering, *Petrasim 2016 user manual*, 2016.

[20] S. S. Gylfadóttir, “Modeling of the námafjall geothermal system: Numerical simulation of response to production and reinjection”, Iceland Geo Survey, Tech. Rep., 2013.

[21] E. Júlíusson, *Personal communication*, Dec. 15, 2016.

[22] C. Kipsang, “Cost model for geothermal wells”, United Nations University: Geothermal Training Program, Tech. Rep., 2013.

[23] H. Moon and S. J. Zarrouk, “Efficiency of geothermal power plants: A worldwide review”, in *New Zealand Geothermal Workshop 2012 Proceedings*, 2012.

Appendix A

Code

The following sections include the scripts of the Algorithm, they are Generator.py, PreProcessor.py, Processor.py, PostProcessor.py

Listing A.1: Generator.py

```python
import os

mname=raw_input('Enter model file name: ')
n=int(raw_input('Number of seperate simulators to generate: '))
sim_name = raw_input('Input name of TOUGH2 exectuion file. Example: t2s_1.exe: ')

r = raw_input('Enter assumed discount rate on an anual basis(number between 0 and 1, not %): ') # discount rate. ex:0.1
price = raw_input('Enter assumed price the operator gets for their power ($/MWh): ') # price of power per MWh in dollars. ex:25
cost_drilling = raw_input('Enter assumed cost of well per meter of depth ($/m): ') # cost of well construction in $/m. ex:2000
h0 = 1000*int(raw_input('Enter assumed enthalpy of geofluid the is reinjected or dumped (kJ/kg): ')) # enthalpy of rejected fluid kJ/kg. ex:1000
eta = raw_input('Enter assumed thermal efficiency of power plant(number between 0 and 1, not %): ')
depth0 = raw_input('Enter the depth coordinates of the surface of the model (m): ')

fh=open('batch.bat','w') # creates and opens batch file
fh.write('python PreProcessor01.py\n') # writes a run command into the batch file for the preprocessor
for i in range(1,n+1):
    fh.write('start python -i Processor'+str(i)+'.py\n')
fh.write('python -i PostProcessor01.py\n')
fh.close()

fin=open('PreProcessor-base.py') # opens the base file for the preprocessor
fout=open('PreProcessor01.py','wt') # creates a preprocessor file for the case that will be simulated
for line in fin:
    fout.write(line.replace('placeholdr',('mname=\''+mname+'\'\n'+'numberof_sims = '+str(n)+'.'))) # relevant variables added into the file
fin.close()

for i in range(1,n+1): #loop that opens the base processor file and creates modified copies for the case that will be simulated
    fname=('Processor'+str(i)+'.py')
    fin=open('Processor-base.py')
    fout=open(fname,'wt')
    for line in fin:
        fout.write(line.replace('placeholdr',('mname=\''+mname+'\'\n'+'sim_num='+str(i)+'\n'+'numberof_sims = '+str(n)+'. \n'+'sim_name=\''+sim_name+'\''))) # relevant variables added into the file
    fout.close()
    fin.close()

fin=open('PostProcessor-base.py') # opens the base file for the postprocessor
fout= open('PostProcessor01.py','wt') # creates a postprocessor file for the case that will be simulated
for line in fin:
    fout.write(line.replace('placeholdr',('mname=\''+mname+'\' \n'+'numberof_sims = float('+str(n)+') \n'+'r = float('+str(r)+') \n'+'price = float('+str(price)+') \n'+'cost_drilling = float('+str(cost_drilling)+') \n'+'h0 = float('+str(h0)+') \n'+'eta = float('+str(eta)+') \n'+'depth0 = float('+str(depth0)+') \n'))) # relevant variables added into the file
fout.close()
fin.close()

os.system('batch.bat') # runs the batch file
```

Listing A.2: PreProcessor.py

```
from mulgrids import *
from t2grids import *
from t2data import *
from t2incons import *
from t2listing import *
from t2thermo import *
from time import *
import os
import shutil

n_years=float(raw_input('Simulation length (years): '))
prod_ind = float(raw_input('Enter asumed productivity index(m^3): ')) #ex: 1e-12

fpath = os.getcwd()+'\\temp\\' #current directory + temp
if not os.path.exists(fpath): # create a temp folder to start moving files into
    os.makedirs(fpath)

```

```
i = int(raw_input('Number␣wellbore␣files:␣'))
wellbores = []
corner_points = []

if i==0:
    corner_points.append([−1e999,−1e999,−1e999,−1e999])
    wellbores.append('DELV')

for n in range(1,i+1):
    while True:
        temp_well = raw_input('Name␣of␣wellbore␣simulator␣file␣nr␣'+str(n)+'(↩
            ↪has␣to␣be␣4␣letters␣or␣numbers␣starting␣with␣f):␣')
        if temp_well[0]=='f' and len(temp_well)==4:
            shutil.copy(temp_well,fpath)
            break
        else:
            print('Not␣a␣valid␣input.␣The␣input␣file␣name␣must␣be␣4␣letters␣or↩
                ↪␣numbers␣and␣start␣with␣f.␣Example:␣f123')
            continue
    wellbores.append(temp_well)
    while True:
        temp_corners = (raw_input('Define␣corner␣points␣on␣the␣surface␣where␣↩
            ↪this␣file␣applies␣(in␣meters.␣format␣should␣be␣x1,x2,y1,y2,depth1,↩
            ↪depth2):␣'))
        temp_corners = temp_corners.split(',')
        x1=float(temp_corners[0])
        x2=float(temp_corners[1])
        y1=float(temp_corners[2])
        y2=float(temp_corners[3])
        z1=float(temp_corners[4])
        z2=float(temp_corners[5])
        if x1==x2:
            print('Not␣a␣valid␣input␣as␣x1=x2.␣Please␣enter␣a␣valid␣input.')
            continue
        elif y1==y2:
            print('Not␣a␣valid␣input␣as␣y1=y2.␣Please␣enter␣a␣valid␣input.')
            continue
        elif z1==z2:
            print('Not␣a␣valid␣input␣as␣z1=z2.␣Please␣enter␣a␣valid␣input.')
            continue
        else:
            break
        if x1>=x2:
            temp_x1 = x2
            x2 = x1
            x1 = temp_x1
        else:
            continue
        if y1>=y2:
            temp_y1 = y2
            y2 = y1
            y1 = temp_y1
        else:
            continue
        if z1>=z2:
            temp_z1 = z2
            z2 = z1
            z1 = temp_z1
        else:
            continue
    corner_points.append([x1,x2,y1,y2,z1,z2])
start_time = clock()

placeholdr # Placeholder for input of model name and number of simulators
#mname='model.dat'
#numberof_sims =

dat = t2data(mname)


dat.parameter['tstop']=n_years*365*24*3600
dat.parameter['tstart']=0*365*24*3600


os.chdir(fpath) #change directory to /temp

dat.write('pytest.txt')
dat = t2data('pytest.txt')
t = 0

blocks = dat.grid.blocklist
num_sims = len(blocks)

for sim_num in range(1,int(numberof_sims)+1): # run copy loop through each of the simulation folders
    folder_name = ('simulator'+str(sim_num))
    i=0
    j=0
    if not os.path.exists(folder_name):
        os.makedirs(folder_name)
    while j<len(wellbores): # run a copy loop through each of the wellbore files
        try:
            shutil.copy(wellbores[j],folder_name)
            print i
            print wellbores[j]
            j=j+1
        except:
            j=j+1
            continue
    i = i+1
print(’starting␣loops␣’+str((clock() − start_time)))

start_time = clock()

for blockn in blocks:
    #if t==0: #loop that excludes the first cell in case that cell is the atmosphere
    # print(’starting write ’+str((clock() − start_time)))
    # fnum = str(’%06.0f’ %(t+1))
    # dat.write(’Input’+fnum)
    # print(’write done ’+str((clock() − start_time)))
    # t=t+1
    # print(’cycle ’+str(t)+’ done ’+str((clock() − start_time)))
    # continue
    print(’________________________________________________________________’)
    print(’cycle␣’+str(t+1)+’␣of␣’+str(num_sims))
    temp_name = str(blockn)
    x = blockn.centre[0]
    y = blockn.centre[1]
    z = blockn.centre[2]
    gen_type=’DELV’
    j=0
    while j<len(wellbores):
        if corner_points[j][0] <= x <= corner_points[j][1] and corner_points[j][2] <= y <= corner_points[j][3] and corner_points[j][4] <= z <= corner_points[j][5]:
            gen_type = wellbores[j]
            break
        else:
            j=j+1
            continue
    gen_name = ’gen␣0’
    gen = t2generator(name = gen_name, block = temp_name, type = gen_type, gx = prod_ind) #Generating cell defined as a deliverability generator
    dat.add_generator(gen)
    print(’starting␣write␣’+str((clock() − start_time)))
    fnum = str(’%06.0f’ %(t+1))
    dat.write(’Input’+fnum)
    print(’write␣done␣’+str((clock() − start_time)))
    dat.delete_generator((temp_name,gen_name))
    t = t+1
    print(’cycle␣’+str(t)+’␣done␣’+str((clock() − start_time)))
    #if t==10: break

end_time = clock()
print(’loop␣done␣’+str((end_time − start_time)))

time_est = len(blocks)*end_time/t
```

```
time_est_hrs = int(time_est/(60*60))
time_est_min = int((time_est-time_est_hrs*60*60)/60)
time_est_sec = time_est-time_est_min*60-time_est_hrs*60*60

print('estimated␣time␣for␣all␣'+str(len(blocks))+'␣blocks␣is␣'+str(time_est_hrs)+'␣hours␣'+str(time_est_min)+'␣minutes␣'+str(time_est_sec)+'␣seconds.')
```

Listing A.3: Processor.py

```
from mulgrids import *
from t2grids import *
from t2data import *
from t2incons import *
from t2listing import *
from t2thermo import *
from time import *
import os
import shutil

placeholdr # placeholder to replace mname, numberof_sims, sim_num ect.
dat = t2data(mname)
#numberof_sims = # number of simulation parts
#sim_num= # simulation part number
#sim_name= # name of tough2 executable file

blocks=dat.grid.blocklist

print('Simulation␣batch␣number␣'+str(sim_num))

fpath=os.getcwd()+'\\temp\\' #current directory + temp
if not os.path.exists(fpath):
    os.makedirs(fpath)
shutil.copy(sim_name,fpath) # copy tough2 to temp folder may require a raw input.
os.chdir(fpath) #change directory
folder_name = ('simulator'+str(sim_num))
if not os.path.exists(folder_name):
    os.makedirs(folder_name)

shutil.copy(sim_name,folder_name) # copy tough2 to temp folder may require a raw input.

n = len(blocks)/(numberof_sims)
#if sim_num==1: n_start=1
#else: n_start = int((sim_num-1)*n)
n_start = int((sim_num-1)*n+1)
n_end = int((sim_num)*n+1)

t=0
for i in range(n_start,n_end):
    fnum=str('%06.0f' %(i))
    shutil.copy(('Input'+fnum),folder_name)
    os.remove('Input'+fnum)
    t=t+1
    #if t==5: break

os.chdir(folder_name)

print(str(n_end−n_start+1) + '␣Simulations␣to␣run')

start_time=clock()

t=0
for i in range(n_start,n_end):
    print('______________________________________________________________')
    print ('Starting␣cycle␣' + str(t+1) + '␣of␣' + str(n_end−n_start+1) + '␣at␣time␣' + str(clock()−start_time))
    fnum=str('%06.0f' %(i))
    fname=('Input'+fnum)
    print ('Reading␣datafile␣' + str(fname) + '␣at␣time␣' + str(clock()−start_time))
    dat=t2data(fname)
    print('Datafile␣read' + '␣at␣time␣' + str(clock()−start_time))
    dat.run(simulator='t2s_1.exe') #dat.run(output_filename=fname+'_out', simulator='t2s_1.exe')
    print('Simulation␣' + fnum + '␣done' + '␣at␣time␣' + str(clock()−start_time))
    t=t+1
    #if t==5: break

print('______________________________________________________________')
print(str(i) + '␣simulations␣successfully␣run' + '␣at␣time␣' + str(clock()−start_time))

end_time=clock()
time_elaps=end_time−start_time
print('Loop␣done␣at␣time␣'+str(time_elaps))

time_est=time_elaps#∗len(blocks)
time_est_hrs=int(time_est/(60∗60))
time_est_min=int((time_est−time_est_hrs∗60∗60)/60)
time_est_sec=time_est−time_est_min∗60−time_est_hrs∗60∗60

print('estimated␣time␣for␣all␣'+str(len(blocks))+'␣blocks␣is␣'+str(time_est_hrs)+'␣hours␣'+str(time_est_min)+'␣minutes␣'+str(time_est_sec)+'␣seconds.')

#os.remove(tough_name)
```

Listing A.4: PostProcessor.py

```
from mulgrids import ∗
from t2grids import *
from t2data import *
from t2incons import *
from t2listing import *
from t2thermo import *
from time import *
import os
import shutil
import time
from operator import itemgetter

placeholdr # place holder to replace mname, numberof_sims, sim_num ect.
#numberof_sims = # number of simulation parts
#mname = #model name
#r = # discount rate
#price = # price of power per MWh in dollars
#cost_drilling = # cost of well construction in $/m
#h0 = # Rejected enthalpy
#eta = # Thermal efficiency of plant
#depth0 = # Depth coordinates of surface

dat = t2data(mname)
blocks=dat.grid.blocklist

rootpath = os.getcwd()
temp_path = os.getcwd()+'\\temp'

start_time = clock()

# copy loop starts
for i in range(1,int(numberof_sims+1)):
    fpath=temp_path+'\\simulator'+str(i) #current directory + temp
    n = len(blocks)/(numberof_sims)
    print(n)
    n_start = int((i-1)*n+1)
    print(n_start)
    n_end = int((i)*n+1)
    print(n_end)
    t=0

    for j in range(n_start,n_end):
        os.chdir(fpath)
        t=t+1
        fnum = str('%06.0f' %(j))
        fnum_next = str('%06.0f' %(j+1))
        fname = 'Input'+fnum+'.listing'
        fname_next = 'Input'+fnum_next+'.listing'
        wait_time = 0
```

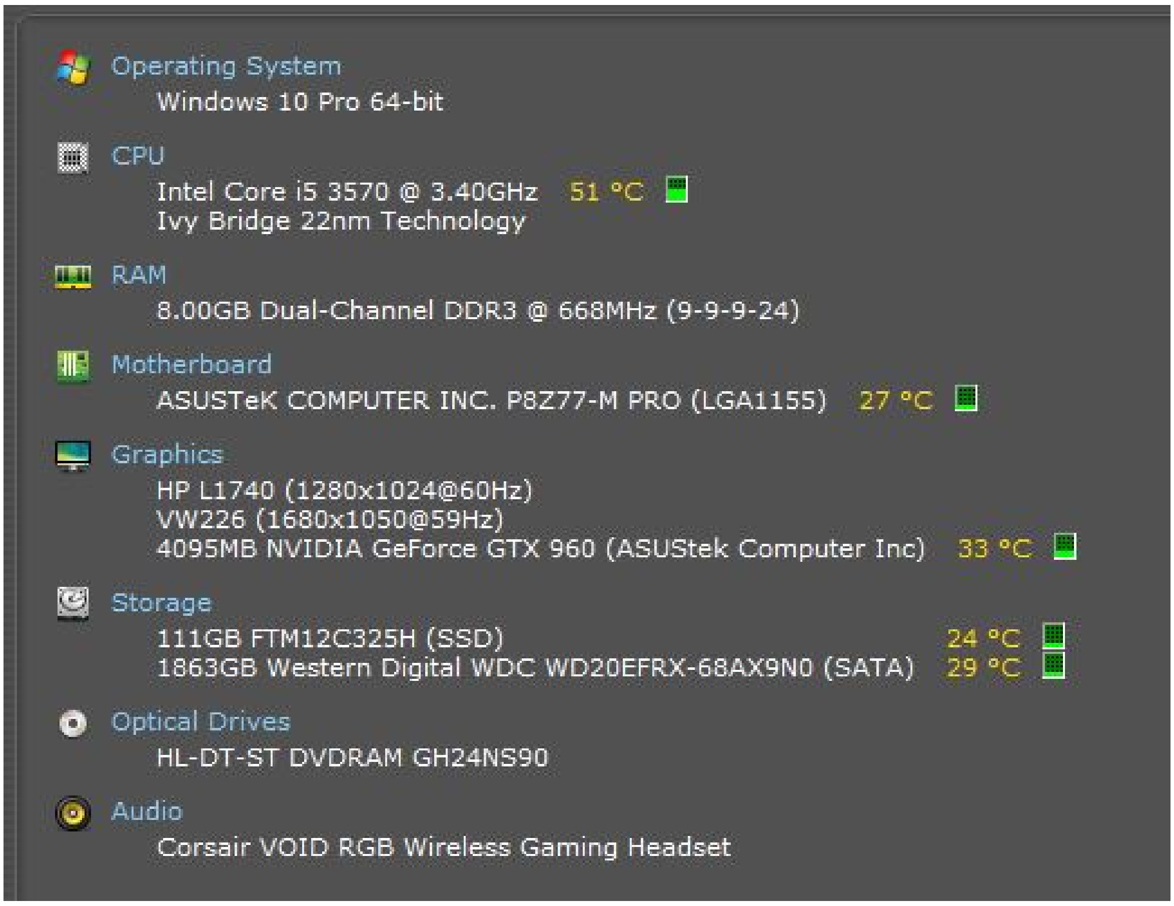
```
print('____________________________________________________________')
switch = 0
if j==n_end-1 and i==int(numberof_sims):
    while switch==0:
        try:
            shutil.copy(fname,temp_path)
            switch=1
            print('File '+fname+' copied at time '+str((clock()-start_time)))
            #os.remove(fname)
            #print('Duplicate deleted')
            break
        except:
            print('Waiting for file '+fname+' to be generated. Total waiting time is '+str(wait_time)+'seconds')
            time.sleep(10)
            wat_time = wait_time+10
elif j==n_end-1:
    print('At end of batch. File number '+str(j))
    path_next = temp_path+'\\simulator'+str(i+1)
    f_next_fold = path_next+'\\'+fname_next # file location in next simulation folder
    while not os.path.exists(f_next_fold) and wait_time<10*60: #wait for file to exist in next folder
        print('Waiting for file '+fname+' to be generated. Total waiting time is '+str(wait_time)+'seconds')
        time.sleep(10)
        wait_time = wait_time+10
    if os.path.exists(fname):
        shutil.copy(fname,temp_path)
        print('File '+fname+' copied at time '+str((clock()-start_time)))
        #os.remove(fname)
        #print('Duplicate deleted')
    else:
        print("Error: %s does not exist, skipping file!" % fname)
else:
    while not os.path.exists(fname_next) and wait_time<10*60: #wait for file to exist
        print('Waiting for file '+fname+' to be generated. Total waiting time is '+str(wait_time)+'seconds')
        time.sleep(10)
        wait_time = wait_time+10
    if os.path.exists(fname_next):
        shutil.copy(fname,temp_path)
        print('File '+fname+' copied at time '+str((clock()-start_time)))
        #os.remove(fname)
        #print('Duplicate deleted')
    else:
```

```
                print("Error:␣%s␣does␣not␣exist,␣skipping␣file!" % fname)
        #if t==5: break
# copy loop ends

# NPV evaluation loop starts
j=0
os.chdir(temp_path)
NPVs = []
for block in blocks:
    fnum = str('%06.0f' %(j+1))
    listing_name = 'Input'+fnum+'.listing'
    try:
        lst=t2listing(listing_name)
        print('Output␣file␣'+listing_name+'␣read␣at␣time␣'+str((clock()-start_time)))
    except:
        j=j+1
        print('Could␣not␣read␣'+listing_name+'␣at␣time␣'+str((clock()-start_time)))
        continue

    try :
        depth = depth0-dat.grid.blocklist[j].centre[2] # depth to center of block in m
        ht,h=lst.history(('g',(str(block),'gen␣0'),'ENTHALPY'))
    except:
        j = j+1
        continue
    ht,h=lst.history(('g',(str(block),'gen␣0'),'ENTHALPY'))
    gt,g=lst.history(('g',(str(block),'gen␣0'),'GENERATION␣RATE'))
    well_cost = depth * cost_drilling # cost of the specific well in $
    NPV = -well_cost
    #NPV = NPV - (h[0]-h0)*g[0]*(365*24/1e6)*price # revenue of year 0 taken out to make a conservative NPV estimate
    for i in range(0,len(gt)):
        t = ht[i]/(365*24*3600) # timestep time in years
        if i==0:
            t_last = 0
        else:
            try:
                t_last = ht[i-1]
            except:
                continue
        P = (-(h[i]-h0)*g[i]*eta)/1e6 # Power output in MW
        try:
            E = P*(ht[i]-t_last)/3600 # Energy output over 1 timestep in MWh
            PV = E*price/((1+r)**(t)) # Present value of revenue in timestep
            NPV = NPV + PV
        except:
            continue
        NPVs.append((str(block),NPV,block.centre,fnum,t))
        j = j+1
        lst.close()
# NPV evaluation loop starts

Sorted_NPVs = sorted(NPVs, key=itemgetter(1), reverse=True)

os.chdir(rootpath)
fhandle = open('NPVs␣of␣wells', 'w')
fhandle.write('Model␣name:␣'+str(mname)+'\n')
fhandle.write('Discount␣rate:␣'+str(r)+'\n')
fhandle.write('Price␣of␣power:␣'+str(price)+'\n')
fhandle.write('Cost␣of␣drilling:␣'+str(cost_drilling)+'\n')
fhandle.write('Rejected␣enthalpy:␣'+str(h0)+'\n')
fhandle.write('Power␣plant␣thermal␣efficiency:␣'+str(eta)+'\n')
fhandle.write('(\'Well␣\',␣NPV,␣array([x␣coordinates,␣y␣coordinates,␣depth␣co-ordinates]),file␣number,simulated␣time)\n')
for item in Sorted_NPVs:
        fhandle.write(str(item)+'\n')
fhandle.close()

end_time = clock()
time_elaps=end_time-start_time
time_elaps_hrs=int(time_elaps/(60*60))
time_elaps_min=int((time_elaps-time_elaps_hrs*60*60)/60)
time_elaps_sec=time_elaps-time_elaps_min*60-time_elaps_hrs*60*60

print('Time␣elapsed␣for␣all␣'+str(len(blocks))+'␣simulations␣is␣'+str(time_elaps_hrs)+'␣hours␣'+str(time_elaps_min)+'␣minutes␣'+str(time_elaps_sec)+'␣seconds.')
```

Appendix B

Computer Specifications

All specifications are acquired through the use of Piriform Speccy version 1.30.730. Information about Speccy can be found on `http://www.piriform.com/speccy`



Operating System
Windows 10 Pro 64-bit
CPU
Intel Core i5 3570 @ 3.40GHz 51 °C
Ivy Bridge 22nm Technology
RAM
8.00GB Dual-Channel DDR3 @ 668MHz (9-9-9-24)
Motherboard
ASUSTeK COMPUTER INC. P8Z77-M PRO (LGA1155) 27 °C
Graphics
HP L1740 (1280x1024@60Hz)
VW226 (1680x1050@59Hz)
4095MB NVIDIA GeForce GTX 960 (ASUStek Computer Inc) 33 °C
Storage
111GB FTM12C325H (SSD) 24 °C
1863GB Western Digital WDC WD20EFRX-68AX9N0 (SATA) 29 °C
Optical Drives
HL-DT-ST DVDRAM GH24NS90
Audio
Corsair VOID RGB Wireless Gaming Headset

Figure B.1: Hardware specifications of the computer used to test the Algorithm.