Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

Amirreza Zamani Meighani

Investigation of the Effect of Uncertain Parameters on a Gas-Lifted Oil Network Using a Non-Linear Model Predictive Control (MPC)

Master's thesis in Chemical Engineering
Supervisor: Johannes Jaschke
Co-supervisor: Evren Mert Turan
January 2022

NTNU
Norwegian University of Science and Technology

Amirreza Zamani Meighani

Investigation of the Effect of Uncertain Parameters on a Gas-Lifted Oil Network Using a Non-Linear Model Predictive Control (MPC)

Master's thesis in Chemical Engineering
Supervisor: Johannes Jaschke
Co-supervisor: Evren Mert Turan
January 2022

Norwegian University of Science and Technology
Faculty of Natural Sciences
Department of Chemical Engineering

NTNU
Norwegian University of
Science and Technology

Investigation of the Effect of Uncertain Parameters on a Gas-Lifted Oil Network Using a Non-Linear Model Predictive Control (MPC)

Department of Chemical Engineering

10/12/2021

Abstract

This work is to find the effect of changing important parameters in a gas lifted oil production network on the production rate and controller behavior. In order to be able to test and experiment the effects, a mathematical model is modelled as a differential-algebraic system of equation and translated into Julia Programming language. Also an economic model predictive controller is made and applied on the system to maximize the profit made in the plant. Moreover, a brief description of the theory behind the work is given. The theory contains different type of optimization problems and respective methods for solving them as well as discretization methods to translate dynamic optimization problem (OCP) in a way that is understandable for numerical solvers.

Having the system working, the model has been tested for the open loop response to examine the stability of the model and several experiments have been made to capture the effect of mentioned parameters on the production rate as well as the NMPC behavior. The experiments can be mentioned as examining the effect of changing the factors such as Gas Oil Ratio (GOR) or productivity index (PI) on the production rate. It is shown and discussed that having higher PI indices in the system would result in more production rate and being more important to NMPC in the optimization problem. Also, for GOR parameter, the system would have less production rate if the GOR parameter decreases in any cases such as aging in the reservoir or being used too much for oil production. In the end the possible future work works are mentioned and the performance of the system is discussed.

Sammendrag

Dette arbeidet er å finne effekten av å endre viktige parametere i et gassløftet oljeproduksjonsnettverk på produksjonshastigheten og kontrollerens oppførsel. For å kunne teste og undersøke effektene, er en matematisk modell modellert som et differensial-algebraisk ligningssystem og oversatt til programmeringsspråket Julia. Også en prediktiv kontroller for en økonomisk modell er laget og brukt på systemet for å maksimere fortjenesten i anlegget. Videre gis en kort beskrivelse av teorien bak arbeidet. Teorien inneholder ulike typer optimaliseringsproblemer og respektive metoder for å løse dem, samt diskretiseringsmetoder for å oversette dynamiske optimaliseringsproblemer (OCP) på en måte som kan løses numerisk.
Etter at systemet fungerer, har modellen blitt testet med åpen sløyfe-respons for å undersøke stabiliteten til modellen, og flere eksperimenter har blitt utført for å fange opp effekten av nevnte parametere på produksjonshastigheten samt NMPC-oppførselen. Forsøkene kan nevnes som å undersøke effekten av å endre faktorene som Gas Oil Ratio (GOR) eller produktivitetsindeks (PI) på produksjonsraten. Det er vist og diskutert at å ha høyere PI-indekser i systemet vil resultere i høyere produksjonshastighet og være viktigere for NMPC i optimaliseringsproblemet. For GOR-parameteren vil systemet også ha mindre produksjonshastighet hvis GOR-parameteren reduseres i alle tilfeller som for eksempel aldring i reservoaret eller blir brukt for mye til oljeproduksjon. Til slutt nevnes mulige fremtidige arbeid og ytelsen til systemet diskuteres.

Contents

Figures

Tables

Code Listings

Chapter 1

Introduction

This work is inspired by the importance of optimal usage of the resources and maximizing the profit in an oil platform since a very small change in production can lead to considerable amount of profit considering an oil production plant in industrial scale.

While small changes in production make considerable difference in profitability in oil production as well as any other business, optimal use of resources become more significant. In order to lessen the manual driven gas injection and production in oil wells, companies introduced automated control techniques Plucenio *et al.* [1]. Information Technology is applied to model, control and optimize the gas lifted operations and this study is focused on using a Non Linear Model Predictive Control (MPC).

1.1 Scope and Objectives

This thesis is a study of uncertain parameters on a gas-lifted oil production network in julia programming language. The scope of this work is to first model the well network containing 3 wells connected to a riser and then apply an MPC framework on the model to maximize the profit of the plant. Establishing the MPC framework, the scope of the work is then to study the uncertain parameter `GOR` and effect of that parameter on the input usage and consequently the profit of the plant. The plant is modelled as an integrator of a system of ordinary differential equations and solved by `BS3` solver. The model inside the optimizer is integrated using direct orthogonal collocation method. The collocation method uses forth order Radau polynomials to solve and integrate the system of differential equations. Note that performing the multi stage scenario based MPC is out of the scope of this thesis while it might be considered as future work using the results of this thesis by the author. The objective of this work can be mentioned as to examine the controller whether it can produce control trajectories for maximizing the profit of the plant and then to see how an uncertain parameter affects on the *objective*.

1.2 Outline of this thesis

Outline of this thesis can be described as below: Chapter 1 describes the gas lift method and the theory behind the work. Also the methods used for optimization as well as how one can formulate an optimization problem. The chapter is finished by introducing the control hierarchy which presents how different control layers are connected together. Chapter 2 aims to describe the gas lifted oil network model and how an MPC framework can be built and maximize the profit of the plant. Furthermore, in Chapter 3, the optimization problem for the model is described to show that how the dynamic problem can be solved using the MPC. Chapter 4 shows the result of applying the controller on the plant as an MPC framework. Also, different situations are studied regarding changing the uncertain parameter and the effect of that on the result. In the end different comparisons and discussion are made to illustrate

1.3 Gas Lift Approach

Obviously, oil and gas reservoirs are the key part in oil and gas industries. In a reservoir, the pressure is the driving force to push the mixture of oil and gas up to the well head. Because the pressure in the reservoir is much greater than the atmospheric pressure so that this pressure difference will affect as the driving force and lead to push the oil from porous media through the well and therefore to the manifold where it can be sent in separators and other upstream equipment. But reservoirs can be depreciated after years of production, thus the pressure inside the reservoir starts to reduce after a time. Passing a certain value for the desired pressure in the reservoir, it would not economically be efficient to use the reservoir for production since lower pressure leads to lower production.

Gas lift technique has been widely used in oil industries to compensate the loss of pressure effect by injecting artificial gas in the well. AS it is studied by Eikrem *et al.* [2], It can be shown that injecting artificial gas close to the bottom of the well can improve production rate by reducing the bottom liquid density. Reducing the density of the liquid in the bottom of the well, it can be moved easier with lower pressure in the reservoir so that the production rate can be improved. To illustrate how a reservoir loses its pressure over the time of production, one can take these points into account: As it is known, crude oil is mixture of many different hydrocarbons which are called different cuts of the crude oil. Starting from Natural gas as one of the lightest to heavier cuts known as bitumen, crude oil can evaporate in almost any temperature and lose some of the matters inside its mixture. Over the time the lighter hydrocarbons evaporate faster than the heavier ones so the density and viscosity of the crude oil changes. That can considerably lower the mobility of the mixture and consequently lower the production rate. Also, when the liquid fraction of the reservoir changes, one important parameter which is called the Gas Oil

Ratio (GOR) would change too. This parameter plays a key role in designing the oil production plant over the reservoir. All in all, one can conclude that passing a period of time using the reservoir for production, the pressure of the reservoir might not satisfy the economics of the oil production plant so that the gas-lift technique may be used in order to improve the production rate. More information on the reservoirs can be found on Satter *et al.* [3]. A schematic of the gas lift technology can be seen in Figure 1.1.

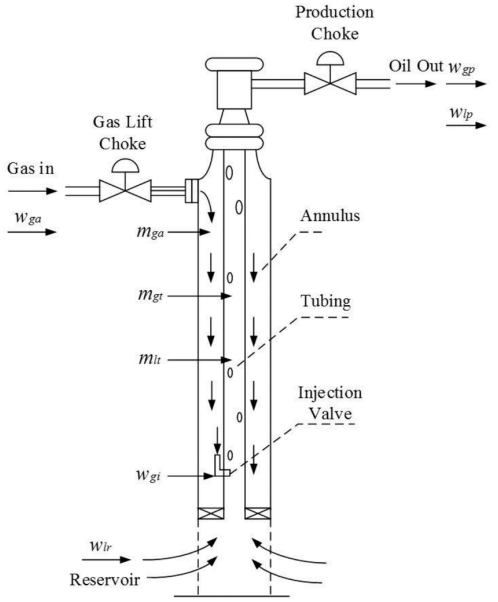


Figure 1.1: Schematic of using the gas lift technique - taken from Hammadih *et al.* [4]

However, it has been studied that the more gas lift injected to the well would not always be helpful. Krishnamoorthy *et al.* [5] studied two wells network system and they showed that injecting too much artificial gas into the system will reduce the production. Therefore there is an optimal value for gas lift which implies the importance of optimization. Figure 1.2 shows the general behaviour of the oil production in front of gas lift. The figure is made by studying the effect of changing the gas lift on the total produced liquid in the plant. The model will be described in the coming parts and different parameters of the model will be introduced.

Looking at the Figure 1.2, one can comprehend that injecting the gas inside the well would immediately increase the produced oil to a certain point but after that, the production rate reduces. So the gas lift might not always be beneficial, but sometimes it can be harmful for the production. The importance of optimization in this case is clear and one can optimize the plant by changing the gas lift injected to the plant. The model and optimization problem is well described in chapters 2 and 3. Also the reason for decreasing the oil production

after a certain value for gas lift is that increasing the gas inside the tubing would increase the friction among the mixture so that the hydro-static pressure drop cannot compensate the new pressure drop caused by friction. When the pressure drop passes the limit, the mixture inside the well cannot be pushed up to the well head so that the production will stop. In this thesis the aim is to study the parameters which can have an effect on the production rate and to set up an optimization problem which will maximize the the profit of the plant where the important cost is the usage of gas lift and income would be produced oil.

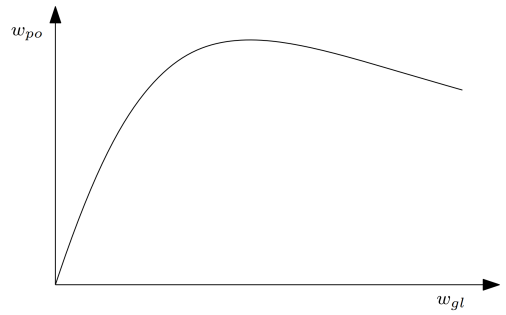


Figure 1.2: Gas lift ω_{gl} and produced oil ω_{po} performance curve

1.4 Introduction to Optimization

Mathematical Optimization or mathematical programming is the procedure of finding the best element regarding some criterion and some sets of available options. In other words, to find the minimum or maximum of a function called *objective function*, in a specific area or over the whole available horizon where the objective function is defined. The optimization problem can or cannot have some constraints which in those cases the optimization problem will be called as constrained or unconstrained programming respectively. Most of the optimization problems have on or more equality and/or inequality constraints.[6]

Equation (1.1) represent the formulation for an optimization problem with the cost function f where ε and Ω are sets of indices for equality and inequality constraints respectively.

$$\underset{x \in \chi}{\text{minimize}} \quad f(x) \tag{1.1a}$$

$$\text{subject to} \quad c_i(x) = 0, \qquad i \in \varepsilon, \tag{1.1b}$$

$$c_i(x) = 0, \qquad i \in \Omega \tag{1.1c}$$

In the above equation, vector x contains all independent decision variable. To have a feasible set of x, all the equality and inequality constraints should be satisfied. Figure 1.3 illustrates the contour of the objective function $f(x)$, also

shows the feasible region which is the set of the points where all constraints are satisfied.

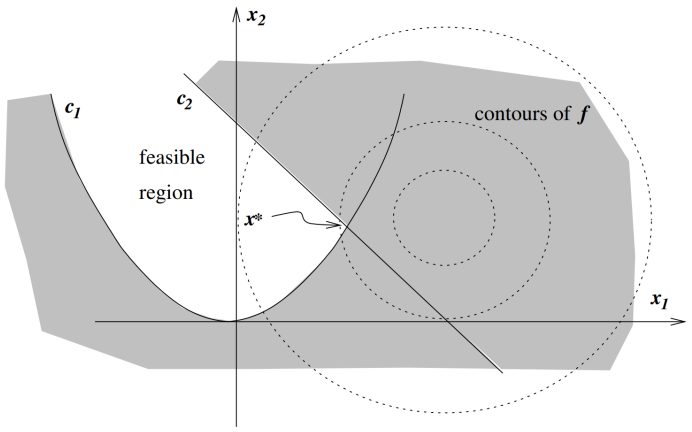


Figure 1.3: Geometrical representation of Equation (1.1), Contour is taken from [6]

1.4.1 Optimization Problems Classification

Optimization problems can be classified by different criterion. Some of the most famous ones are listed below.

Constrained or unconstrained optimization: As it was said later, the optimization problem can have one or more equality and/or inequality constraints. For example the Equation (1.1) is a constrained optimization problem since the objective function is limited by sets of equality and inequality constraints. The objective function *f(x)* can freely be distributed without any conditions which makes an example of unconstrained optimization problem like what is shown in Equation (1.2).

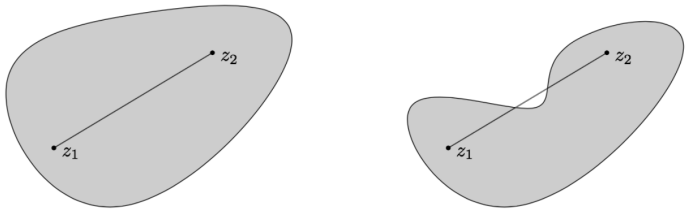
$$\min_{x \in \chi} \quad f(x) \tag{1.2}$$

Local or global optimization: Optimization problems can also be classified by the type of solutions they have or people look for. In most cases, solver look for local optimal solution which is a point where the objective function take its minimum or maximum value compared to a neighbourhood around that point. In convex programming or more specifically speaking, in linear programming, a local solution is the global solution to the optimization problem while non-linear optimization problems(both constrained or unconstrained) might take local solutions which are not globally an optima. In most of the cases recognizing a global solution would be difficult and expensive and locating that point would be even more expensive in time and computation costs.

Convex or non-convex optimization: Convexity in optimization is an essential concept. Many of the optimization problems have this feature which makes it much easier to solve compared to the situation where an objective function does not possess the convexity feature. Both functions and sets can b *convex*. A set $S \in R^n$ is convex if the straight line which connects any two elements of the set is entirely inside the set or mathematically speaking, for two elements x and y inside the set S, one should prove that $\alpha x + (1-\alpha)y \in S$, for all $\alpha \in [0,1]$. The same principle is found for a function f aver the convex domain S, if for two points x and y from S the following property is satisfied:

$$f(\alpha x + (1-\alpha)y) \leq \alpha f(x) + (1-\alpha)f(y), \text{ for all } \alpha \in [0,1] \tag{1.3}$$

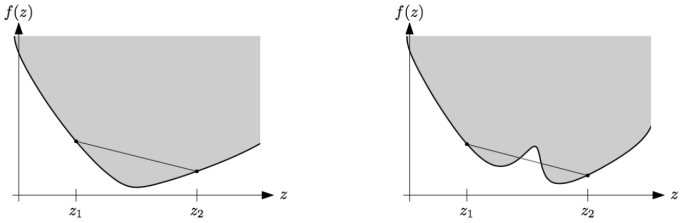
Considering the terms mentioned above, an optimization problem is called convex if the objective function f(x) is convex and the feasible set x is also convex. Figure 1.4 shows two arbitrary sets which are convex (a) and non-convex (b)



(a) A convex set. (b) A nonconvex set.

Figure 1.4: An illustration of convex and non-convex arbitrary sets [7]

Having a convex optimization problem, one can find the global solution by only finding the local solution to problem since in a convex optimization problem, a local optimum is guaranteed to be the global optimum. Note that if either the objective function or the feasible set is non-convex there would be no guarantee for local solution to be the global solution. Figure x represent the schematic of a convex and a non-convex functions.



(a) A convex function. (b) A nonconvex function.

Figure 1.5: An illustration of convex (a) and non-convex (b) arbitrary functions [7]

1.4.2 Different Types of Programming

Taking a closer look into the objective function, optimization problems can be categorized based on the different types of objective functions they have. The objective function can be linear, quadratic, non-linear or so many other forms. But for some specific types of objective function, there are special formulation of optimization problem where specific solvers can handle those problems easier. A brief explanation to some of the most famous types of optimization problems (programming) comes in the rest of this section.

Linear Programming (LP): If the objective function *f(x)* is linear and all the constraints are linear too, the optimization problem is then called linear programming (LP). A linear programming can be written as what is shown in Equation (1.4).

$$\begin{aligned} \min_{x \in \chi} \quad & \mathrm{d}^{\mathrm{T}}(x) \\ \text{subject to} \quad & g_i(x) = a_i^T x - b_i = 0, \qquad i = 1, \ldots, m, \\ & h_i(x) = a_j^T x - b_j \leq 0, \qquad i = 1, \ldots, n, \\ & x_{\min} \leq x \leq x_{\max} \end{aligned} \tag{1.4}$$

Where d is a vector containing the known coefficient.

Quadratic Programming (QP) Taking the Equation (1.4) and substitute the objective function to a quadratic function, the optimization problem will be called quadratic programming (QP). A quadratic programming is an optimization problem where the objective function is quadratic and all the constraints are linear. This can be formulated as what is shown in Equation (1.5).

$$\begin{aligned} \min_{x \in \chi} \quad & \mathrm{x}^{\mathrm{T}}\mathrm{Q}\mathrm{x} + \mathrm{d}^{\mathrm{T}}\mathrm{x} \\ \text{subject to} \quad & g_i(x) = a_i^T x - b_i = 0, \qquad i = 1, \ldots, m, \\ & h_i(x) = a_j^T x - b_j \leq 0, \qquad i = 1, \ldots, n, \\ & x_{\min} \leq x \leq x_{\max} \end{aligned} \tag{1.5}$$

Quadratic programmings can be either convex or non-convex based on the form of quadratic objective function. In Equation (1.5), the optimization problem is convex if the n-dimensional matrix Q is positive semi definite ($Q \geq 0$). $Q = 0$ is also by definition, positive semi definite.

Non-Linear Programming (NLP): If any of the constraints or objective function in non-linear, the optimization problem would take the form of a non-linear optimization problem or in short *NLP*. NLPs can be written in the same way as Equation (1.1) despite that the objective function and/or constraints are non-linear.

1.5 Solving Algorithms

There are different ways to build a solver to solve an optimization problem which can be explicit or iterative in term of solving. There might be an explicit method for solving a very simple optimization problem while for more complicated problems, finding an explicit way might cost too much in time and computation and sometimes it is impossible to find an explicit method for solving optimization problems such as non linear programmings. On the other side, iterative algorithms can be designed in a way to tackle the problem iteratively and solve them until a stopping criteria is satisfied. The general approach for an iterative method to optimization problems(OPs) is shown below:

Algorithm 1 Algorithm 1 Iterative solution procedure of OPs

Given initial point x0 and stopping criteria

While stopping criteria not fulfilled **do**
  Compute the next iteration point
end while

Table 1.1: Iterative Algorithm to solve OPs [7]

As it is stated in the algorithm, an initial point *x0* should be given to the method. Some algorithms need a feasible initial points while some others, like sequential quadratic programming (SQP), does not require a feasible starting point. Also, stopping criterion could be set via different measures such as:

- When the number of iterations reaches the limit
- Via gradient matrices such as gradient matrix of objective functoion ∇f, or the Lagrange of the objective function$\nabla \mathcal{L}$.
- A posturing of the optimal point. Like when the gradient matrices are smaller than a defined value ϵ

Also regarding optimization problems with inequality constraints, one can advice the *Interior Point Methods (IP)*. This method penalize the inequality constraints using a barrier function and solves the inequality constraints as equalities. A barrier function in mathematics is a continuous function where value on a point would rise to infinity as the point closes to the boundary of the feasible region of the OP.

1.6 Optimal Control

Dynamic systems are characterized by evolving in time. That means the variable are not only functions of space, but also are functions of time while, on the contrary, one can find static systems independent of time. There are different way to optimize dynamic systems and in this thesis, only the *Dynamic*

optimization will be considered and other approaches such as *Quasi Dynamic Optimization* is neglected. Dynamic optimization, as it was mentioned before is to optimize a dynamic system which means to find a time dependant solution for an optimization problem with time variant decision variable.

Optimal control plays a key role in connecting the optimization and control where the dynamic systems can be controlled by optimization.In other words, optimal control is to find the proper inputs into the system in a way that all the constraints are satisfied. In order to do so, one should be able to discretize the dynamic system in predictive states so that the solver can apply an iterative algorithm on the problem and solve it. Note that in this thesis the focus will be on the continuous dynamic systems. An example of continuous in time optimal control problem is given in Equation (1.6).

$$\underset{z,\,u}{\text{minimize}} \quad E(z(t)) + \int_0^T L(z(t), u(t))dt \tag{1.6a}$$

$$\text{subject to} \quad z(0) - z_0 = 0 \quad , \tag{1.6b}$$

$$\dot{z}(t) - f(z(t), u(t)) = 0, \quad t \in [0, T], \tag{1.6c}$$

$$g(z(t), u(t)) = 0, \quad t \in [0, T], \tag{1.6d}$$

$$h(z(t), u(t)) \leq 0, \quad t \in [0, T], \tag{1.6e}$$

$$r(z(T)) \leq 0 \tag{1.6f}$$

Taking Equation (1.6) into account, one can explain that in the objective function mentioned in OCP, the integral cost contribution $L(z, u)$ is often called *Lagrange term* which should not be confused with lagrange function. Also the first term $E(z(t))$ which is the terminal term is sometimes referred to as *Mayer term*. An objective function which is mixed from these two terms is called *Bozla objective* [8].

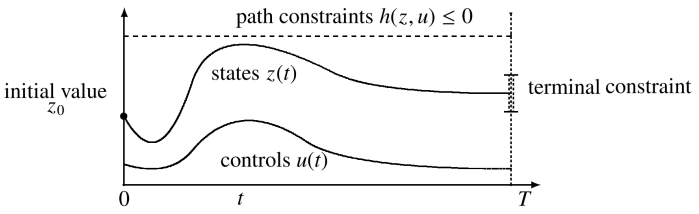


Figure 1.6: The constraints and of a continuous in time optimal control problem (OCP)

Figure 1.6 illustrate the optimal control problem and its variable. In Equation (1.6), $t \in [0, T]$ denotes the time vector, $z(t) \in \mathbb{R}^{n_z}$ represents the the vector (matrix) of state decision variables and $u(t) \in \mathbb{R}^{n_u}$ is for control decision variables (inputs). Moreover, the constraint Equation (1.6b) sets the initial value fixed and subequation Equation (1.6c) illustrates the ODE system

where all the differential equations are defined. Path constraints are shown in Equation (1.6e) and the last set of constraints which is shown in Equation (1.6f) are called terminal constraints. [8]

1.7 Methods for Solving OCPs

Generally speaking, there are three common approaches to tackle the continuous optimal control problems (OCPs) which can be mentioned as *state space*, *direct* and *indirect* approaches. In this section , an introduction to these methods is given.

State-space methods usually use the optimality basis which say that all subarcs of an optimal trajectory should be optimal by itself. This rule leads to the *Hamilton-Jacobi-Bellman (HJB)* equation which is a partially differential equation in state space. There are some methods, numerically written to calculate the solution approximations but, the problem is that mostly they suffer from what is called *Bellmans "curse of dimensionality"* which is limited by small state dimensions.

Also, there is another category among the methods for solving a continuous in time OCPs which is called *Indirect Methods*. Creating a boundary value problem (BVP) using the optimality conditions is the key point in these methods. The rule in this category is to *first optimize ,then discretize* so the optimality conditions are expressed in continuous form and then discretized to form a numerical solution. Non-linearity could be a strong barrier for these methods since it costs a lot in computation and time. Figure 1.7 summarize different categories for solving an optimal control problem.

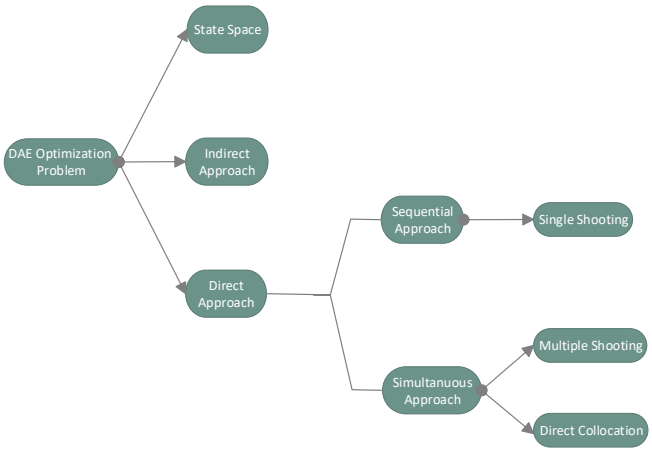


Figure 1.7: Different methods to solve optimal control problems (OCPs) [9]

1.7.1 Direct Methods for Solving OCPs

Without need to derive a BVP, direct methods are able to solve optimal control problems by transforming the problem into a finite *Non-Linear Problem (NLP)*. So these types of methods follow the methodology *first discretize, then optimize*. Direct methods can handle all types of constraint since they are being treated by most of the well developed NLP methods which work with active changes [8]. Also, they can deal with both ordinary differential equation (ODEs) and differential-algebraic system of equations (DAEs) [10]. This group can be broken down into two sub-categories, sequential and simultaneous. These methods are illustrated in Figure 1.7 and elaborated below.

Direct single shooting: As the simplest idea for solving an OCP directly, single shooting method follows this idea: reducing the boundary value problem to a initial value problem *IVP* and solving the IVP using a numerical integrating fashion. As an example of how this method works, one can consider the BVP problem shown in Equation (1.7).

$$\begin{aligned} &\frac{d^2y}{dx^2} = f(x, y, \frac{dy}{dx}) \\ &y(a) = y_a, \quad y(b) = y_b \end{aligned} \tag{1.7}$$

As it is clear the problem is a boundary value problem. One can say that they want to reduce it to an initial value problem where they can keep the initial condition $y(a) = y_a$ and guess the initial condition for the first gradient value $\frac{dy}{dx}(a) = Y_a$ and use a numerical integration method to discretize the problem and integrate the problem to find the boundary value $y(b) = \lambda_b$. Then the only step left is to compare the λ_b with y_b and use a trial and error method to find a $\frac{dy}{dx}(a)$ such that $\lambda_b = y_b$.

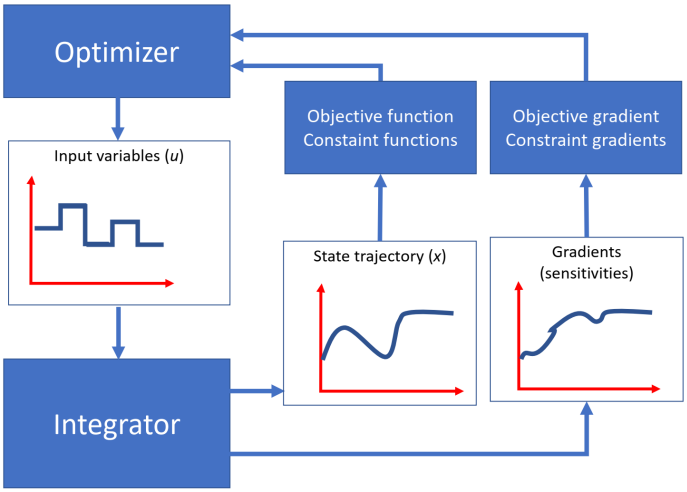


Figure 1.8: Schematic illustration of Direct single shooting Method [9]

The word *shooting* is used since the initial guess has to be guessed. Ap-

plying the method on an OCP, one can say that the single shooting in OCP solving is to take the initial value and find the optimal trajectory such that all the constraints are satisfied. In other words only the control parameter(u) is discretized and will be the NLP decision variable and not the states. A scheme of this method is shown in Figure 1.8.

Direct multiple shooting: In single shooting, the integration is performed on the whole time horizon $[t_0, t_f]$ which might be impotent because of too long period for integration. The idea in multiple shooting is to break the integration period down to finite number of smaller time intervals and include the states x_k in the NLP. The idea originally comes from Osborne [11] in it is to divide the time interval $[0, T]$ into shooting intervals $[t_k, t_{k+1}] \subset [0, T]$ and most often the shooting intervals are uniform. Then the integration is performed on each different interval with initial guesses for each (S_i). Afterwards the whole problem can be imported into an NLP solver where a new set of conditions which is called *continuity conditions* is added to the problem to produce a continuous state trajectory. This is shown in Figure 1.9.

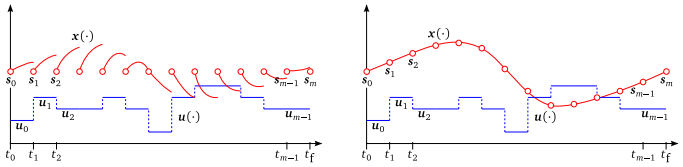


Figure 1.9: Schematic illustration of Direct multiple shooting Method (DMS) discretization applied to the optimal problem. On the left, initialized shooting nodes where solution is breaking constraints. On the right, the constraints are satisfied after convergence of NLP[12]

The continuity condition which is added to the NLP can be described as a constraint which sets all the small intervals uniform to create a continuous state trajectory. It can be shown as what is given in Equation (1.8):

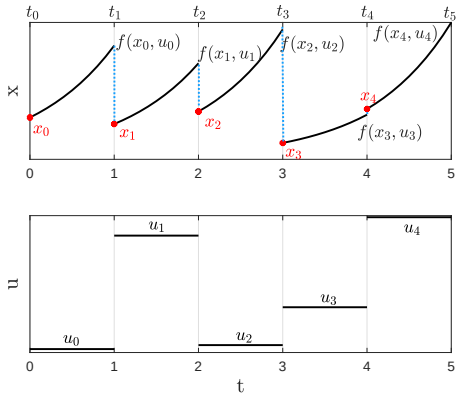


Figure 1.10: Schematic illustration of Shooting Gap in DMS Method

$$x_{k+1} - f(x_k; u_k) = 0, \forall k \in 1; ...; N \tag{1.8}$$

The multiple shooting method results in producing much more variables in NLP compared to the number of variables in single shooting which makes a larger optimization problem but since the integration periods are smaller than the integration period in single shooting, the solver reaches the convergence faster.

As a comparison b between single, and multiple shooting method, one can note that both of the methods rely on separate integrator package and optimized integration routines can be applied on both single and multiple shooting methods. Single shooting is conceptually simple since it is based on discretizing only the control input U which results in small optimization problem ($n_u \times N$ variable) while multiple shooting is conceptually more complicated and results in larger optimization problem $(n_u + n_x) \times N$ variables. Regarding nonlinearity, single shooting can be unsuitable in unstable or highly nonlinear problems. On the other hand, the multiple shooting method can perfectly handle unstable and non-linear problems. Constraints are enforced at check points in single shooting while multiple shooting method enforce the constraints for states at each sample time [9].

Direct collocation: As is was mentioned in previous parts, the shooting based methods put an integrator (mostly numerical ones) to integrate the differential equations in each step points and import them to a solver so that the optimization problem can be solved. Biegler [13] proposed the concept of letting the solver (optimizer) also handle the integration. This is the philosophy behind the orthogonal collocation method.

As it was mentioned, in collocation method, the solver takes care of integration as well as solving the optimization problem. This is done by approximating the solution of the differential equation by an order of K polynomial.To start explaining the method, one can consider Equation (1.9) which represents a differential equation.

$$\dot{Z} = f(z) \tag{1.9}$$

The Equation (1.9) can be approximated by a third order polynomial like what is shown in Figure 1.11and Equation Equation (1.10).

$$Z(t) \approx A + Bt + Ct^2 + \frac{1}{3}Dt^3 \tag{1.10}$$

As it is obvious, there should be at least K+1 points to determine an order K^{th} polynomial. Figure 1.11 illustrate a third order polynomial which approximate the function f in a time interval $[t_0, t_f]$. Neglecting the proof and transforming the equation Equation (1.10) in matrix form, one can show that using the orthogonal collocation would result in having the solution as what is shown in Equation (1.11) where M represents a matrix of constants which

can differ based on the choice of different lagrange polynomial i.e., Radau, Legandre or so on.

$$\begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \\ \dots \end{bmatrix} = \begin{bmatrix} Z_0 \\ Z_0 \\ Z_0 \\ \dots \end{bmatrix} + M \begin{bmatrix} \dot{Z}_1 \\ \dot{Z}_2 \\ \dot{Z}_3 \\ \dots \end{bmatrix} \tag{1.11}$$

Once one prepares the matrix M, they can "translate" the system of differential equations into an acceptable form for optimizer. Note that this method would results in increasing the size of optimization problem considerably. In other words, this method can trade nonlinearity with size of the optimization problem by introducing collocation points.

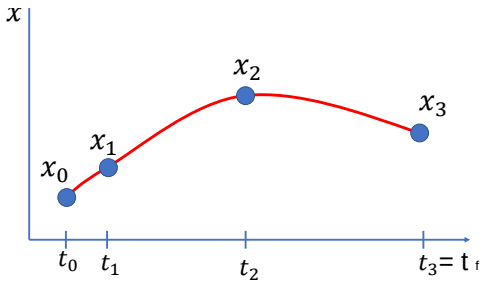


Figure 1.11: Schematic illustration of approximated polynomial with intermediate collocation points

Like the other numerical methods, there would be a big error if the method is applied on a "too large" interval say the whole horizon where the problem is defined. One solution to this problem could be introducing finite elements throughout the horizon and applying the collocation method on each element which would considerably increase the accuracy Biegler [13]. The trade off between having more collocation points or finite elements has been the matter of discussion while academia accepts the idea of increasing the number of finite elements better.

1.8 PLantwide Control

Plantwide control regards the structural decisions which is involved in designing control system of a chemical plant. More specifically, plantwide control is concerned with the questions "which variables should be controlled, measured, manipulated and what links should be made between them?" [14]. The goal is to find sets of variables which when are kept in certain values, lead the plant to be operated near the optimal conditions. Because different parts of a large scale system have different time scale, there should be different layers for controlling the system.

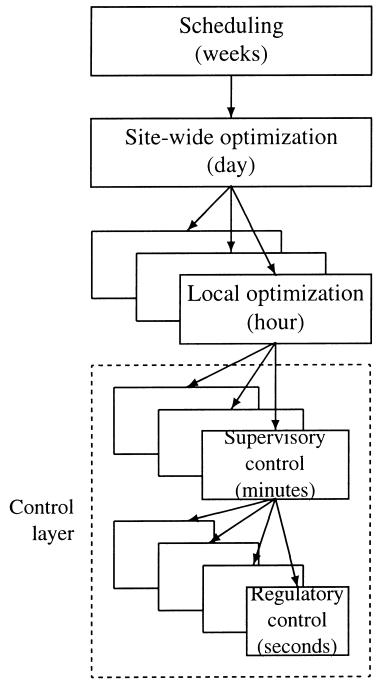


Figure 1.12: Schematic illustration of different layers (time scales) in plantwide control

Figure 1.12 briefly represents these layers combining the selection of control structures in chemical plants. Considering large scale chemical systems like multi-well oil production plant it is required to have a stable control structure. Two main objectives are to achieve long term economical objectives and short term stability in the system. Based on plantwide control rules, the control structure includes (*i*) scheduling (weeks), (*ii*) site-wide (real-time) optimization (days), (*iii*) local optimization (hours), (*iv*) Supervisory or Model predictive control (MPC, minutes), and finally (*v*) stabilizing and regulatory control (seconds). Layers (ii) and (iii) compromise the economic aspects of the system while the supervisory and regulatory layers are concerned with setpoint tracking from the layers above (setpoints for regulatory layers are given from higher layers). So, these layers are interdependent through the controlled variables. Note that the supervisory layer is a slow economic layer which tries to satisfy the economics of the system using the control variables which are high worth. On the other hand, this layer produces the setpoints for regulatory layers which deal with those control variable that have less effect on the economics of the plant. The economic MPC can be noted as on of the means to be used in supervisory layer.

Chapter 2

Model Description

This chapter describes how a model for a network of wells in an oil production plant can be written to imitate the behavior of the actual plant. To recap the previous the actual plant, the model should describe the gas lift effect on the production rate and encompass the parameters in the actual system. To keep the derivation as straightforward as possible, the model derivation starts with writing the mass balance equations for a gas lifted oil production plant with one well and in the end of this chapter, the differential-algebraic system of equations for a multi well gas lifted oil production network is given.

2.1 Modelling of Gas Lifted Wells

In this section, a brief description of a gas lifted well model is given. The model is used in the optimization problem as a network of wells which can be combination of several wells with different characteristics. The derivation can be started with a model for one gas lifted well and the way to include all wells as a model in optimization problem. A schematic representation of a network of gas lifted wells in an oil production plant is presented in Figure 2.1. Note that the model for the gas lifted well as well as the DAE system formulations are taken from the work done by Krishnamoorthy *et al.* [15].

For a gas lifted well model, the model derivation consists of four main parts: (*i*) mass balance in the various phases; (*ii*) density models; (*iii*) pressure models and (*iv*) flow models. Mass balance in one well can be written as Equation (2.1) where the variables are defined as what is shown in table Table 2.1.

$$\dot{m}_{ga} = \omega_{gl} - \omega_{iv} \tag{2.1a}$$

$$\dot{m}_{gt} = \omega_{iv} - \omega_{pg} + \omega_{rg} \tag{2.1b}$$

$$\dot{m}_{ot} = \omega_{ro} - \omega_{po} \tag{2.1c}$$

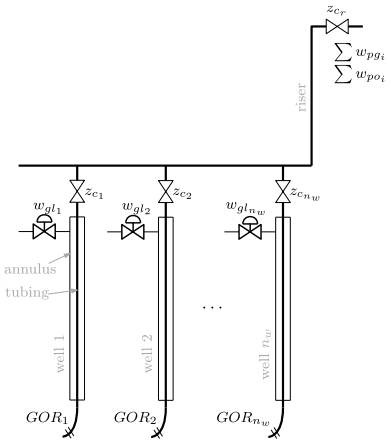


Figure 2.1: Schematic illustration of an oil production plant with three gas lifted wells. Each well has different characteristic and the gas lift ($\sum \omega_{gl}$) is injected to the wells to improve the production rate ($\sum \omega_{po_i}$) [16]

The equations mentioned in Equation (2.1a) to 2.1c are the three main differential equations in the model which describe the behavior of the mass in annulus and well tubing. The rest of equations will express the algebraic part of the model. The pressure distribution can be expressed by equations (2.2a) to (2.2d). where P_a is annulus pressure, P_{wi} is well injection point pressure, P_{wh} is well head pressure and the bottom hole pressure is P_{bh}.

$$P_a = \left(\frac{T_a R}{V_a M_w} + \frac{g L_a}{L_a A_a} \right) m_{ga} \tag{2.2a}$$

$$P_{wh} = \frac{T_{wi} R}{M_w} \left(\frac{m_{gt}}{L_w A_w + L_{bh} A_{bh} - \frac{m_{ot}}{\rho_0}} \right) \tag{2.2b}$$

$$P_{wi} = P_{wh} + \frac{g}{L_w A_w} \left(m_{ot} + m_{gt} - \rho_0 L_{bh} A_{bh} \right) H_w + \Delta P_{fric} \tag{2.2c}$$

$$P_{bh} = P_{wi} + \rho_w g H_{bh} + \Delta P_{fric} \tag{2.2d}$$

Also, the density of gas in the annulus (ρ_a) and density of fluid mixture in tubing (ρ_m) can be found through the equations written in Equation (2.3).

$$\rho_a = \frac{M_w P_a}{T_a R} \tag{2.3a}$$

$$\rho_m = \frac{m_{gt} + m_{ot} - \rho_o L_{bh} A_{bh}}{L_w A_w} \tag{2.3b}$$

Where the molecular weight is represented as M_w, annulus temperature as T_a, gas constant as R, Reservoir oil density as ρ_o, cross-sectional area of the

Name	Definition
m_{ga}	Mass of the gas in the annulus
m_{gt}	Mass of the gas in well tubing
m_{ot}	Mass of the oil in well tubing
ω_{gl}	Rate of gas lift injection
ω_{iv}	Gas flow from annulus into the tubing
ω_{pg}	Produced gas flow rate
ω_{po}	Produced oil flow rate
ω_{rg}	Gas flow rate from the reservoir
ω_{ro}	Oil flow rate from the reservoir

Table 2.1: Variable definition in mass balance equations shown in Equation (2.1)

well below and above the injection point as A_w and A_r respectively and well length below and above the injection point as L_w and L_r respectively. Also, the flow rates through different parts of the system can be written as following:

$$\omega_{iv} = C_{iv}\sqrt{\max(0, \rho_a(P_a - P_{wi}))} \tag{2.4a}$$

$$\omega_{pc} = C_{pc}\sqrt{\max(0, \rho_{wi}(P_{wi} - P_m))} \tag{2.4b}$$

$$\omega_{pg} = \frac{m_{gt}}{m_{gt} + m_{ot}}\omega_{pc} \tag{2.4c}$$

$$\omega_{po} = \frac{m_{ot}}{m_{gt} + m_{ot}}\omega_{pc} \tag{2.4d}$$

$$\omega_{ro} = PI(P_r - P_{bh}) \tag{2.4e}$$

$$\omega_{rg} = GOR \cdot \omega_{ro} \tag{2.4f}$$

Using the Diameters of the annulus and the tubing, D_a and D_w, the respective cross-sectional areas can be computed. A_a and L_a are the cross-sectional area and length of the annulus respectively, T_w is the well tubing temperature, L_a is the annulus length, H_w and H_r are the well tubing vertical height above and below the injection point, respectively, T_w is the well tubing temperature, and g is the gravity acceleration constant. Also in equation set (2.4), valve flow coefficients for the production choke and the downhole injection valve are represented as C_{pc} and C_{iv} respectively ; moreover, Pr is the pressure in the reservoir, PI is the productivity index of the reservoir, Pm is the manifold pressure and GOR is the gas–oil ratio. Note that no pressure coupling can be found between the wells in the mentioned formulation.

Considering the mentioned equations, one can realize that all the three differential equations and the algebraic ones are coupled to each other so that they can form a Differential-Algebraic system of equations (DAE) which is

described in the next section. Among the parameters in the model, some can have uncertainty with them which make the problem more complicated and one might have to apply methods for optimization with uncertainty. In this thesis, all the parameters in optimization problem are assumed to have a certain value and only the effect of changing the uncertain parameter on different aspects of the plant is considered.

2.2 DAE System for the Plant

As it can be seen from the derived model in Section 2.1, the model is a semi-explicit index-1 DAE (differential-algebraic system of equation) of the form:

$$\begin{aligned} &\dot{x}_i = f_i(x_i, z_i, u_i, p_i) \\ &g_i(x_i, z_i, u_i, p_i) = 0, \quad \forall i \in \mathcal{N} = \{1, ..., n_w\} \end{aligned} \tag{2.5}$$

Where f_i is the set of differential equations mentioned in Equation (2.1) and g_i is the set containing all the algebraic equations mentioned in (2.2a) to (2.4f). Also the subscript i refers to each well in a network of n_w wells. Again, note that the subscript i has been removed from the model equations in Section 2.1 for more convenience. The variables and parameters presented in Equation (2.5) are differential states (x_i), algebraic states (z_i), decision variables (u_i), and the parameters (p_i) which can be expressed as:

$$x_i = [m_{ga_i},\ m_{gt_i},\ m_{ot_i}]^T \tag{2.6a}$$
$$z_i = [\rho_a,\ \rho_m,\ P_{a_i},\ P_{wh_i},\ P_{wi_i},\ P_{bh_i},\ \omega_{iv_i}, \omega_{pc_i},\ \omega_{pg_i},\ \omega_{po_i}]^T \tag{2.6b}$$
$$u_i = [\omega_{gl_i}]^T \tag{2.6c}$$
$$p_i = [GOR_i]^T \tag{2.6d}$$

Considering a network of wells consisting of n_w wells, the system of equations can be written as Equation (2.7)

$$\dot{x} = f(x, z, u, p) \tag{2.7a}$$
$$g(x, z, u, p) = 0 \tag{2.7b}$$

Where x, z, u, and p are the matrices containing the values for all wells. For example, x is the matrix of all x_i where $i \in 1, ..., n_w$. The combined states, control input and parameters are shown in more detail in equation4.

$$x = [x_1^T, \quad x_2^T \quad ... \quad x_{n_w}^T]^T \tag{2.8a}$$
$$z = [z_1^T, \quad z_2^T \quad ... \quad z_{n_w}^T]^T \tag{2.8b}$$
$$u = [u_1^T, \quad u_2^T \quad ... \quad u_{n_w}^T]^T \tag{2.8c}$$
$$p = [p_1^T, \quad p_2^T \quad ... \quad p_{n_w}^T]^T \tag{2.8d}$$

The resulting model is a differential-algebraic system of equations with three differential and 12 algebraic equations all coupled to each other. The input for the model is the gas lift flowrate and the output is the total amount of produced oil. Note that for the sake of simplicity, the models mentioned above can be easily transformed into an ordinary differential equations (ODE) by eliminating the algebraic variables and creating three differential equations.

2.3 Software Package

For modelling purpose, the Julia programming language was used. The model was written as a function file which takes the input of the model and the time which user wants to integrate the model for, and then the model will integrate and sole the model based on the input and the required time. Note that the gas lifted well model can be used either for one well or for a network of gas lifted wells by introducing the number of wells (Nw) as another input to the model. For simplicity, the DAE system is transformed to an ordinary differential system of equations and by taking the initial values and the time span for integration, the function will solve the problem using (BS3) solver in julia. Afterwards, Algebraic states can be computed using the results of solver and the parameters given to the problem.

Chapter 3

Problem Formulation

An introduction to optimal control problems (*OCPs*)was given in Section 1.6. One can understand from that section that an optimal control trajectory would be the result of solving the OCP and one can use it to keep the system at its optimal conditions. But that would be slightly risky because of the probability of deviation in model with respect to the actual plant and in that case, the computed input trajectory would not be beneficial anymore. [8] and [6] Moreover, there might be external factors affecting the plant behaviour and not considered in the model and consequently using the given model might not lead to the points that the actual system produces. Considering all of these matters, one might consider a way to "monitor" the model in order to tackle the deviation or unwanted disturbances during the time that simulation is working. Having the means to monitor the model, it would be possible to modify the input trajectory or some parts of the optimal control problem such as relaxing a condition by a relaxation factor and so on. And by doing that, the simulated model can always be kept as similar as the actual system and even more, it can be kept as close as desired optimal condition. For instance if the gas to oil ratio for one of the wells is changed, the controller (optimizer) should be able to first see that change and second it should be able to give a new optimal input trajectory to the plant in a way that the optimal conditions are satisfied. Monitoring the system and act based on the changes is also called "*Feedback Control*".

This chapter aims to present a constructive way to fill the gap between the optimization and control by introducing Model Predictive Control (*MPC*) framework. Furthermore, this chapter demonstrates the approach used to construct a nonlinear program using the collocation method, as well as the software package utilized to solve this huge NMPC system.

3.1 MPC Framework

Considering the real plant working over the time, some parts of the characteristics of the plant, i.e. *GOR* might be affected by disturbances. Moreover,

measurement also can be subject to noises or measurement errors. So, one can conclude that feedback control or monitoring the model and change the decision periodically would be more effective [17]. As it can be guessed by the name, The model predictive control (MPC) uses a model to predict the future to decide for control input and produce an optimal input trajectory for a prediction horizon of $T = t_f - t_0$ where t_0 is the starting point and t_f is the final time. In addition, to be able to have a reliable MPC, one should provide not a perfect, but a considerably accurate model for the MPC [18]. The procedure of making decision for each step (as long as prediction horizon, T) of simulation time and repeating the decision making until reaching the final time for simulation horizon is called Model Predictive Control (MPC). Note that prediction horizon is normally much shorter than simulation horizon.

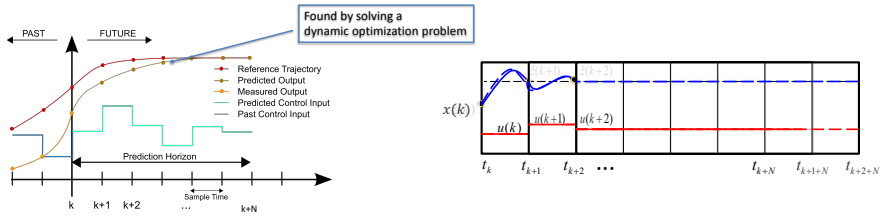


Figure 3.1: MPC procedure on the left and the order of computed optimal input trajectory for each time step

As it can be seen from the Figure 3.1, MPC uses the past data from the model and predicts the future as long as the prediction horizon is by solving a dynamic optimization problem (OCP). One might use the simulation horizon as prediction horizon and solve the OCP only once if they have the perfect model. In most cases, finding the perfect model would be too expensive in time and computation energy so that, an effective idea is to break the simulation horizon into smaller pieces and solve the OCP several times. Each step would be called as prediction horizon. Having the optimal trajectory by solving the OCP for the time period of prediction horizon, one can use the first element of that trajectory (first sample time, i.e. second or minute) and shift the prediction horizon for the next turn and repeat the procedure until the end of simulation horizon. This can be summarized in four main steps [17]:

- Take the system's present state $\tilde{x}_0$
- Use the state $\tilde{x}_0$ predict the future by solving the optimization problem for N steps called prediction horizon. 3. Apply the first part of resulted optimal control trajectory u_{opt} to the plant.
- Apply the first part of resulted optimal control trajectory u_{opt} to the plant.
- Take the new state from the system and shift the optimization horizon one step forward and repeat the procedure.

The last step in the MPC is referred to as moving horizon control, because

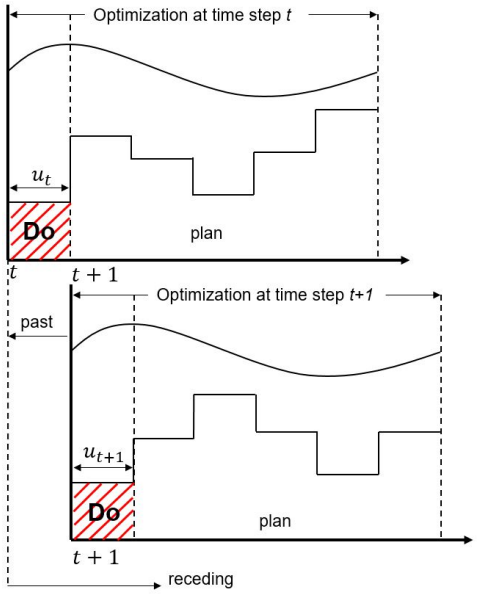


Figure 3.2: Moving horizon strategy shown for the first two steps [17]

of the allotment of the prediction time horizon T. See Figure 3.2 to see the moving horizon strategy. Also, figure 3.3 illustrates the general procedure for MPC to solve the OCP Nocedal and Wright [6].

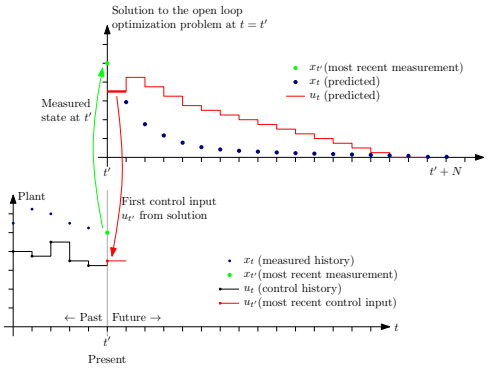


Figure 4.1: Illustration of the MPC principle.

Figure 3.3: Illustration of the MPC principle [6]

The main benefit of model predictive control over single-step optimization is that it combines feedback control with open loop optimization by demanding a new solution from the open loop optimization based on future observed errors, as seen in the algorithm below.

There are different choices for the objective function in MPC where it can be minimization of the error between states and references or directly controlling the profit of the plant. The first case would make a (setpoint)

Algorithm 2: State feedback MPC procedure

for t = 0,1,2,3,... **do**
  Get the current state value from measurement data;
  Solve the dynamic OCP on the prediction horizon T from t to t+N;
  Apply the first control action u_t on the plant;
end for

tracking MPC and the second case would result in having an Economic MPC or Dynamic Real-time Optimization (DRO) which would have the objective function in the form of what is shown in Equation (3.1). Hence the Economic MPC. In this thesis, the goal is to control the gas lift well network plant with an economic MPC.

$$Min\left(\sum_{k=0}^{k=N_{c_1}} \cos t(x_k, u_k) + \sum_{k=0}^{k=N_{c_2}} (u_k - u_{k-1})R(u_k - u_{k-1})\right) \tag{3.1}$$

Having the mentioned equation in 3.1, it should be noted that the coefficient R should be as small as possible, but large enough to give a stable controller. In other words, all the concern is on the cost function while the controller is stable. The objective function for the well network plant is explained in the next sections.

3.2 Nonlinear MPC

Considering the huge loads of constraints and stiff specification on the gas lifted oil production plant as well as considerable as well as demanding economical surveillance, one can find it highly non-linear. Having the plant as a non-linear model, the controller turns to be non-linear too because it has the model inside itself and it will be called as a *Nonlinear Model Predictive Control, (NMPC)*. The NMPC would have the same basics and rules as what is shown in Figure 3.3 but the model would be non-linear.

As it was mentioned in 2.2, one can write the gas lifted oil network as a differential-algebraic system of equations in the following form:

$$\begin{aligned} \dot{x} &= f(x, z, u, p) \\ g(x, z, u, p) &= 0 \end{aligned} \tag{3.2}$$

Where the states and control input are assumed to be measurable. The revenue of the plant is computed by the total amount of oil produced ($\sum_{i=0}^{n_w} \omega_{po_i}$) and the main cost is the gas lift usage. So, an NMPC like what is shown in Equation (3.3) can be written to maximize the profit of the plant.

$$\begin{aligned}
\text{minimize} \quad & -\sum_{t=0}^{N-1} J_{profit} + \gamma \sum_{t=0}^{N-1} \|\Delta u\|_2 && && \text{(3.3a)}\\
\text{subject to} \quad & x - x_0 = 0 && , && \text{(3.3b)}\\
& \dot{x}(t) - f\left(x(t), z(t), u(t), p\right) = 0, && t \in [t_0, t_f], && \text{(3.3c)}\\
& g\left(x(t), z(t), u(t), p\right) = 0, && t \in [t_0, t_f], && \text{(3.3d)}\\
& \sum_{i=1}^{n_w} \omega_{pg_i} \leq \omega_{pg_{\max}} && , && \text{(3.3e)}\\
& x^{low} \leq x(t) \leq x^{high} && , && \text{(3.3f)}\\
& z^{low} \leq z(t) \leq z^{high} && , && \text{(3.3g)}\\
& u^{low} \leq u(t) \leq u^{high} && , && \text{(3.3h)}\\
& \Delta u^{low} \leq \Delta u(t) \leq \Delta u^{high} && && \text{(3.3i)}
\end{aligned}$$

Where the profit function and Δu are given in the following form:

$$J_{profit} = \alpha_{po} \sum_{i=1}^{n_w} \omega_{po_i} - \alpha_{gl} \sum_{i=1}^{n_w} \omega_{gl_i} \tag{3.4}$$

$$\Delta u_t = u_t - u_{t-1} \tag{3.5}$$

Note that in Equation (3.3), $x, z, uandp$ are differential and algebraic states, control input and parameters or characteristics of the wells. Parameters are assumed to be constant during the time of simulation. Also, the objective function in 3.3 implies the maximization of the costs in the plant while the minimum amount of gas lift is used and all the constraints are satisfied. The cost function also contains total oil produced in the whole system as revenue and the gas lift usage as cost. α_{po} and α_{po} are the prices for oil and gas lift respectively.

The bounds such as upper and lower bounds are considered for differential, algebraic and control input variables in order to prevent different parts of the system from having too much or too less regarding their respective values. Moreover, higher and lower bounds are defined for control input change regarding the time (Δu). This would prevent the control input trajectory from having big jumps and give a better trajectory in case of smoothness; However, physical characteristics of valves in the plant might not handle immense changes and because of that, the control trajectory should be smooth to certain levels.

Note that optimization problem 3.3 does not have any feedback control. Hence it illustrates an open loop optimization a non-linear DAE. The loop can be closed by using the MPC fashion like what is written below:

Algorithm 3: State feedback MPC procedure

for t = 0,1,2,3,. . . ,N-1 **do**
  Get the current state value from measurement data;
  Solve the dynamic OCP on the prediction horizon T from t to t+N;
  Apply the first control action u_t on the plant;
end for

3.3 Descretization of the OCP

The basics for direct collocation method was explained in Section 1.7.1. Since the solvers used for solving optimization problems cannot handle the dynamic optimization problems directly, one should use a method for translating the dynamic optimization problem (OCP) such that the solver can take the problem and solve it to find the optimal input trajectory. The mentioned method would be discretization of the DAE system and transforming the optimal control problem. The discretization of the gas lifted well network is done using the direct orthogonal collocation [13]. This section aims to present a process of discretization using the direct collocation method as well as transcription of the gas lifted well network OCP.

Considering the OCP states and control variable, one can take the collocation method and discretize them all on the collocation interval $t \in [t_k, t_{k+1}] \subseteq [t_0, t_f]$ where $k \in [0, ...0, N-1]$, using a Kth order polynomial l. The polynomial l is a function of coefficient $\tau \in [0, 1]$. The formulation which results in having the polynomial using a set of collocation points such as shifted Gauss–Legendre or Radau is given below:

$$\ell_k(\tau) = \prod_{k=0,\neq j}^{K} \frac{\tau - \tau_k}{\tau_j - \tau_k} \tag{3.6}$$

Where the Table 3.1 can be used to find the coefficient τ based on the order of the polynomial and type of the collocation points.

Also based on what is given in Biegler [13], the collocation and continuity equation are given as below respectively:

Collocation equation:

$$\sum_{j=0}^{K} x_{ij} \frac{d\ell_j(\tau_k)}{d\tau} = h_i f\left(x_{ik}, z_{ik}, u_k, p, t_{ik}\right), \quad k = 1, ..., K \tag{3.7}$$

Continuity equation:

$$x_{i+1,0} = \sum_{j=0}^{K} \ell_j(1) z_{ij} \tag{3.8}$$

Neglecting the proof (see [13]) section 10.2 for proof), the differential and

Degree K	Legendre Roots	Radau Roots
1	0.500000	1.000000
2	0.211325	0.333333
	0.788675	1.000000
3	0.112702	0.155051
	0.500000	0.644949
	0.887298	1.000000
4	0.069432	0.088588
	0.330009	0.409467
	0.669991	0.787659
	0.930568	1.000000
5	0.046910	0.057104
	0.230765	0.276843
	0.500000	0.583590
	0.769235	0.860240
	0.953090	1.000000

Table 3.1: Collocation points for two different types of lagrangian polynomial coefficients, shifted Gauss–Legendre and Radau roots

algebraic states (x , and z respectively) can be discretized using the collocation method in the following form:

$$x_k(t) = \sum_{j=0}^{K} \ell_j(\tau) x_{ij} \tag{3.9a}$$

$$z_k(t) = \sum_{j=0}^{K} \ell_j(\tau) z_{ij} \tag{3.9b}$$

Note that the control inputs are assumed to be piece-wise constant in the interval $[t_k; t_{k+1}]$, which is given by:

$$u(t) = u_k \tag{3.10}$$

Summing up, the OCP is descretized into a finite dimensional nonlinear programming broken into N uniformly spaced sample intervals in $\mathcal{K} = \{1, \cdots, N\}$. This is done using the third order Radau collocation points which creates an approximation of the system mentioned in Equation (3.2) as shown in Figure 3.4. The reason for choosing Radau collocation points is that the Radau collocation selects a set of collocation points that includes the interval's end point. This is a beneficial trait for stiff systems [6].

Note that the index $c \in \mathcal{C} = 0, 1, 2, 3$ is used to show three collocation points and their initial state in each intervals [k, k+1]. Applying the mentioned

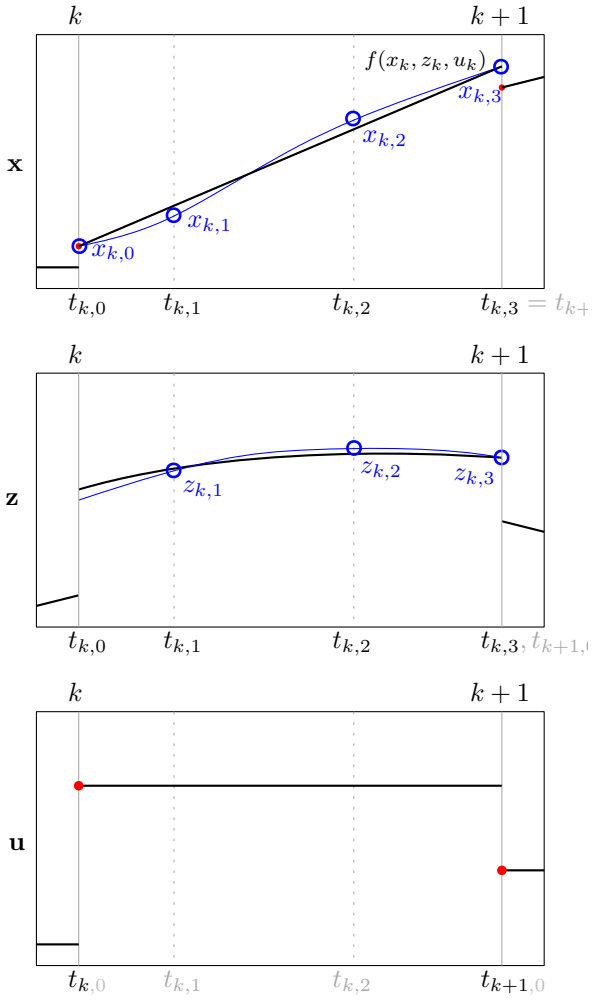


Figure 3.4: Schematic illustration of third order orthogonal direct collocation method using Radau pattern for positioning the collocation points. The figure is representing the approximation of the dynamic system mentioned in 2.5 for an interval $[k, k+1]$. Note that by adding one extra collocation point for differential states at $t_{k,0}$, the shooting gap would be enforced to be zero and the continuity of the states would be ensured. Also the control input u is constant over each interval $[k, k+1]$ [15].

technique, the discretized differential states $\tilde{x} = (x_{k,c}|k \in \mathcal{K}, c \in \mathcal{C})$ and the algebraic ones $\tilde{z} = (z_{k,c}|k \in \mathcal{K}, c \in \mathcal{C})$ would then take the format shown below:

$$\tilde{x} = [\ x_{1,1}^T \quad x_{1,2}^T \quad x_{1,3}^T \quad x_{2,1}^T \quad \cdots \quad x_{N-1,3}^T \quad x_{N,1}^T \quad x_{N,2}^T \quad x_{N,3}^T\ ]^T \tag{3.11a}$$

$$\tilde{z} = [\ z_{1,1}^T \quad z_{1,2}^T \quad z_{1,3}^T \quad z_{2,1}^T \quad \cdots \quad z_{N-1,3}^T \quad z_{N,1}^T \quad z_{N,2}^T \quad z_{N,3}^T\ ]^T \tag{3.11b}$$

The combined states for n_w wells in Equation (3.11a) at time instant k and the collocation point c in the interval $[k, k+1]$ are represented by $x_{k,c}$. The final state variables $x_{k,3}$ and the beginning conditions of the following time interval x_0 must be equivalent to maintain state continuity across two successive time intervals, where the vector of initial states at each interval is represented by:

$$x^0 = [\ x_{1,0}^T \quad x_{1,0}^T \quad \cdots \quad x_{N,0}^T \quad x_{N+1,0}^T\ ]^T \tag{3.12}$$

Moreover, as it is mentioned in 3.4, the discretized control inputs $\tilde{u} = (u_k|k \in \mathcal{K})$ are expected to be piecewise constant during each sample interval and so are not discretized at the collocation points. This is shown in Equation (3.13). Also, the parameters p in the system are assumed to be invariant by time.

$$\tilde{u} = [\ u_1^T \quad u_2^T \quad \cdots \quad u_N^T\ ]^T \tag{3.13}$$

The discretized system dynamics system at any time piece k can also be written as:

$$F(\tilde{x}_k, x_k^0, \tilde{z}_k, \tilde{u}_k, p) = 0 \tag{3.14}$$

The daily production optimization problem can be framed as a standard NLP problem with N evenly spaced sampling intervals in $\mathcal{K} = \{1, \cdots, N\}$ on a prediction horizon from $k = 1$ to $k = N$ once the system has been discretized [15]. The following equations yield the vector of decision variables for the NLP problem across this prediction horizon, as well as the discretized NLP:

$$\theta = \left[\ \cdots \quad \underbrace{x_{k,0}^T}_{x_k^0} \quad \underbrace{x_{K,1}^T \quad \cdots \quad x_{K,3}^T}_{\tilde{x}_k} \quad \underbrace{z_{K,1}^T \quad \cdots \quad z_{K,3}^T}_{\tilde{z}_k} \quad \underbrace{u_K^T}_{\tilde{u}_k} \quad \cdots\ \right]^T \tag{3.15}$$

$$\underset{\theta}{\text{minimize}} \quad J = -\sum_{k=1}^{N} J_{profit} + \gamma \sum_{k=1}^{N} \|\Delta u\|_2 \tag{3.16a}$$

$$\text{subject to} \quad F(\tilde{x}_k, x_k^0, \tilde{z}_k, \tilde{u}_k, p) = 0, \quad \forall k \in \mathcal{K}. \forall p \in \mathcal{U}, \tag{3.16b}$$

$$\sum_{i=1}^{n_w} \omega_{pg_i} \leq \omega_{g_{\max}}, \quad \forall i \in \mathcal{N}, \tag{3.16c}$$

$$x^{low} \leq x_{k,c} \leq x^{high}, \quad \forall k \in \mathcal{K}. \forall c \in \mathcal{C}, \tag{3.16d}$$

$$z^{low} \leq z_{k,c} \leq z^{high}, \quad \forall k \in \mathcal{K}. \forall c \in \mathcal{C}, \tag{3.16e}$$

$$u^{low} \leq u_{k,c} \leq u^{high}, \quad \forall k \in \mathcal{K}, \tag{3.16f}$$

$$\Delta u^{low} \leq \Delta u_k \leq \Delta u^{high}, \quad \forall k \in \mathcal{K}, \tag{3.16g}$$

$$x_{k,3} = x_{k+1}^0, \quad \forall k \in \mathcal{K}, \tag{3.16h}$$

$$x_{1,0} = x_0 \tag{3.16i}$$

The objective function is constituted of the economic cost function which is stated Equation (3.4) as well as the tuning parameter γ, which penalizes the control effort. Equation (3.16c) is used to implement the total gas capacity constraints, where $\omega_{g_{\max}}$ is the maximum gas capacity. The state constraints Equation (3.16b) is used to create the discretized dynamic model. Upper and lower bound constraints on differential and algebraic states, as well as upper and lower bound constraints on decision variables, are applied at each collocation point and sample time, as demonstrated in Equations 3.16d to 3.16f. Equation (3.16g) implements rate of change constraints on the decision variables. Equation (3.16h) implements the shooting gap restrictions to maintain state continuity. Equation (3.16i) enforces the initial conditions. When it comes to constrained optimization, the best approach is one with active gas capacity constraint.

3.4 NMPC Framework

Throughout this thesis, a brief description of the whole plant and the equations and dynamic system approximated to imitate the actual plant was given. In Section 3.3, the optimal control problem (OCP), which was designed to control the plant at the highest accessible profit, has been discretized to a minor finite-dimensional non-linear optimization problem (NLP) to be able to be fed into a non-linear solver. The problem has been turned to closed loop using the MPC algorithm in order to manage (unpredicted) external factors to the plant and control the plant in the best way.

From now, the model should be translated into an understandable computer language and then it can be run as a simulation. This thesis is done using the Julia programming language. Julia is a high-level, high-performance

dynamic programming language that has been designed to contain general programming purposes while having lots of well-suited tools for computational science and numerical analysis. See Julia documentation for more details *Julia documentation user manual* [19].

The enormous sized optimization problems can be coded and performed in Julia with considerable speed in computation using the pre-loaded packages, which can be used for defining the optimization problem, i.e. JuMP, (see JuMP documentation [20]) or non-linear solvers such as IPopt [21]. This work is coded in Julia using three main scripts and two auxiliary scripts all connected to each other to simulate the actual plant using the model given in Chapter 2, set up the optimizer to take the states from the model and return the optimal input trajectory and a simulator to imitate the MPC fashion (closed-loop). Auxiliary files are helping the three main scripts, designed to produce the M matrix for collocation method (see 1.11), prepare and return the suitable set of parameters depending on the number of wells or compute the initial steady states optimizer and model. The NMPC is often used to provide set-points to a lower regulatory layer, such as PI controllers (see Figure 1.12), which operate the valves to maintain the plant operating at an optimal steady state. On the other hand, the regulatory layer is not taken into account in this thesis. Instead, the NMPC is employed to directly control the valves in the gas-lifted oil network. To be more specified, the procedure of finding the optimal gas lift trajectory for the gas lifted oil network can be described as below:

1. The steady-state values of x, z and u are given to the problem and assumed to be optimal (planning the RTO is not considered in this thesis).

2. The NMPC takes the provided initial points for the states and control input and solves the NLP to find the optimal input trajectory over the prediction horizon. Afterwards, the NMPC gives the plant the first step of the optimal trajectory.

3. The plant runs and solves the DAE system to find the states for the next step and return them to the NMPC.

4. The simulator manages all these data transforms between the files to simulate the closed-loop simulation for the system. It is also in charge of data recording and creating proper matrices containing all differential and algebraic states and optimal control sequences.

The results of the work is presented in the next chapter where each of them has been discussed separately. Also, a brief description of the possible future work is given in the end of next chapter.

Chapter 4

Results and Discussion

This chapter aims to illustrate the result of the work which has been done in this thesis. Throughout the thesis, the model has been depicted and different parts of it have been described. Also the optimization problem which can be used in solver is introduced. The programming part of the work has been done using the Julia programming language and the setup is used for studying different aspects of the work, such as studying the effect of changing the key parameters on the objective function or testing the NMPC files package in order to see if it can handle the situation where a constraint changes. Briefly mentioning, these studies are considered in this chapter:

- Open loop part
  - Open loop response of the system
  - Effect of Gas Lift on Production Rate
- Closed loop part
  - Effect of Parameter GOR on Production Rate
  - Effect of Parameter PI on Production Rate
  - Effect of Changing the Maximum Gas Constraint on Production Rate

The work starts with finding the steady states of the plant which can be found using he model and initial points given to the problem such as initial points for three differential states and the initial value for gas lift. Once the steady states of the plant are found, the optimizer would start to solve the NLP to produce the optimal input (gas lift) trajectory. Then the first part of the input sequence is fed into the plant model file which solves the ODE system to find the states for the next time step. Afterwards, the states would be taken back to the optimizer to close the loop and the procedure will be repeated until the time reached the simulation time. In the codes configuration it has been tried to make everything general so that the user can change the parameters as an input to the function. Some of the most important ones of these parameters can be mentioned as:

- Number of the wells to make it more sufficient for further studies where the system can have more or less than three wells.
- Several time intervals such as the prediction horizon for optimizer, simulation time, the integration time for the plant model so that the simulation can be done in different ways
- For more information, see the codes in appendices

Solving the NLP in optimizer has been done using the JuMP package in Julia which makes defining the problem considerably easier since the problem can be defined in multi dimensional matrices so that all the differential states containing the interior points (collocation point), data for each wells of the information of the states on each finite elements can be defined in one matrix. Once the optimization problem is defined using the JuMP, it can be solved using the IPopt plugin which takes the NLP and returns the optimal input trajectory. Furthermore, the ODE system which is an approximation of the actual oil production plant is solved using the *BS3()* solver.

4.1 Open Loop Simulation

Setting the NMPC program up, one can start the numerical studies by checking the open loop simulation of the plant. This can be done using the plant file which is basically an integrator that can solve the ODE system and find the states for the next time step. The open loop simulation is done using a step change on the gas lift flow rate which is the input of the plant. The open loop simulation is done when the wells have different reservoir pressure so that the stability of the system can be tested. The result of the test is illustrated in Figure 4.1 where it shows the total oil production trajectory (the subplot in the middle) as well as total gas produced (the subplot in the bottom) and the gas lift flow rate which has a step change.

Symbol	Description	Well 1	Well 2	Well 3	units
GOR	Gas oil ratio	0.1	0.1	0.1	-
PI	Productivity index	1.2	2.2	3.2	Kg/Pa/S
P_{res}	Reservoir pressure	140	150	160	bar

Table 4.1: Parameters used in open-loop step response on gas injection valves for wells with varying reservoir pressure

The step change test is done using the initial gas lift and differential states values for each well and a jump in the gas lift flow rate to the 60% of the maximum value of the gas lift (when the valves are fully open). Figure 4.1 shows that the system reaches the steady states after around 6 minutes. Also the effect of increasing the gas lift is visible where increasing the gas lift flow rate increase the total produced gas in the system. It also results in increasing

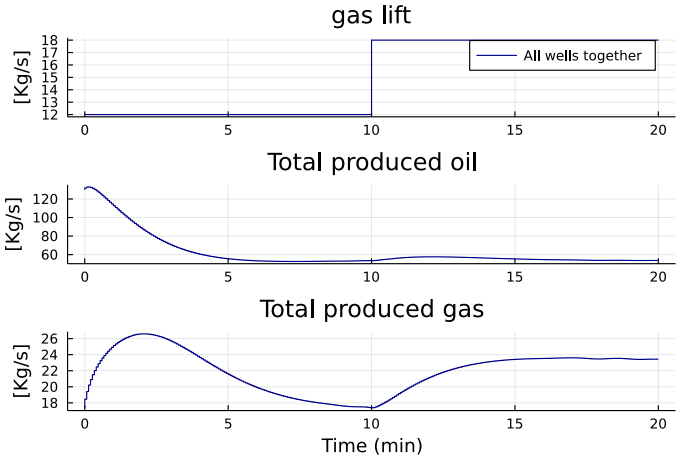


Figure 4.1: The open loop step change test is represented by the three shown plots. On the top, gas lift flow rate is shown which has a jump form 4 [Kg/s] to 6 [Kg/s]. In the middle, the total produced oil represented and the in the last subplot in the bottom, the total produced gas is illustrated.

the total produced oil but accumulation of the gas inside the wells would have negative effects on the production rates. This is explained more deeply in the next sections. All in all, one can conclude that the resulting model is stable and it responds to the changes correctly. Some of the important parameters used as wells characteristics are shown in Table 4.1.

4.2 Effect of Gas Lift on Production Rate (in Open Loop)

Testing the model in open loop, one can examine the response of the system to changes in system input. In the modelled oil production network, the gas lift flow rate is the only input to the system which can affect the production rate considerably. In this part it is tried to show the fact that increasing the gas lift is not always beneficial for the plant and raising the flow rate too much would cost a lot for the plant since it would result in less production rate. Figure <XX> is showing the effect of increasing the gas lift flow rate to the wells with respect to the total oil produced. The gas lift is increased from 4 [Kg/s] to 8 [Kg/s] and the total produced oil from the plant is recorded. As it can be seen from the figure, after a certain value for gas lift flow rate, it would not be beneficial for the production rate to increase the gas lift more than before. Figure 4.2 shows that the production rate falls down after around 5.5 [Kg/s] of gas lift injected to the plant. The reason for this effect is that increasing the gas lift inside the well would reduce the pressure difference between the reservoir and bottom hole of the well since it is the driving force for the mass inside the well to be able to go up along the the well to the riser and finally to

the manifold. The curve is explained by the hydro-static pressure drop, which is insufficient to compensate for the increased friction loss caused by the rise in gas mass in the well tubing.

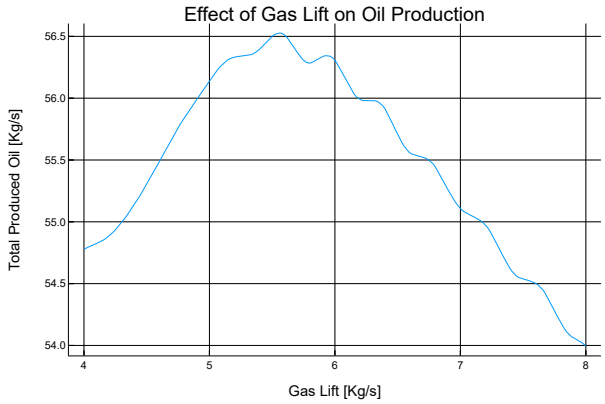


Figure 4.2: Increasing the gas lift flow rate with respect to the total oil production is represented. On the horizontal axis, the flow rate for gas lift in each well is depicted while the vertical axis illustrates the total oil production flow rate

It can also be shown that by increasing the gas lift, the driving force for the mass inside the wells which is the pressure difference $\Delta P = \mathrm{P}_{res} - P_{bh}$ would have the same behavior as the total oil produced. In fact the reason for dropping the production rate is that, after a certain value, increasing the gas lift would only increase the bottom hole pressure which results in reducing the pressure drop between the reservoir and bottom hole of the well and consequently having less driving force would kill the production rate. Altogether, there is always an optimum for gas lift flow rate and it cannot be increased forever since it would not be advantageous for the system

4.3 Closed Loop Simulation

In the real processes, the steady state cannot be achieved forever and the processes run in dynamic state. Passing the open loop tests for the system, one can close the loop by adding the feedback effect. The close loop in oil production network is achieved by introducing the NMPC framework and simulate the real plant by using the MPC algorithm mention in previous parts. The first study in closed-loop part is to test the controller if it can solve the NLP and maximize the profit of the plant (see 3.4). The plant ran for a simulation time of 10 minutes to see the NMPC behavior. Also the bounds for the NLP is applied to the optimization problem using a 4^{th} order collocation method. The

maximum possible gas into the plant (see 3.16c)is set to 28 [Kg/s]. The results are shown in Figure 4.3 where the results for each well separately (on the left) and summation of all three wells (on the right) are visible.

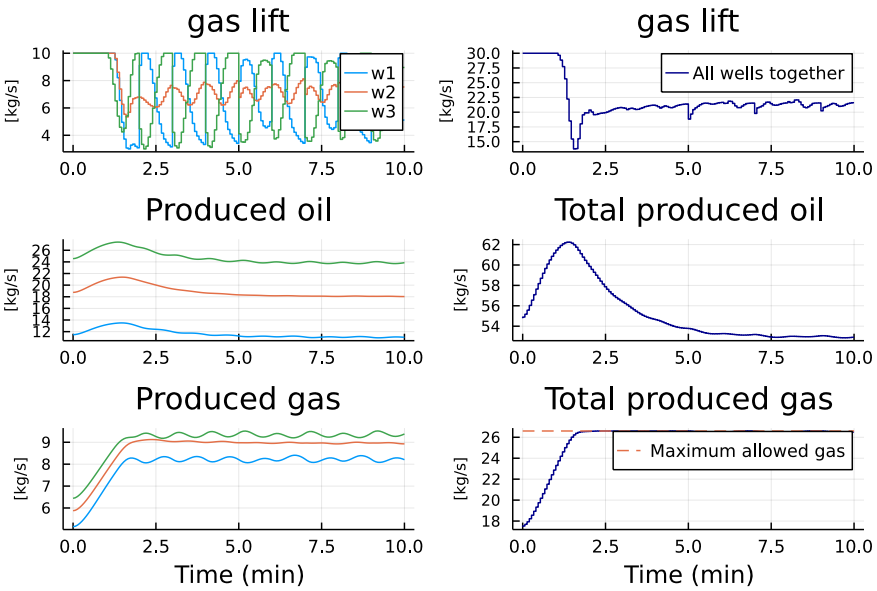


Figure 4.3: The closed loop response of the NMPC to the oil production well network. Left side has the data for each well separately naming w1, w2 and w3. On the right, the data for summation of three wells can be found. The maximum allowed flow rate for total gas lift usage and produced gas are 30 and 28 [Kg/s] respectively

Taking a look at the Figure 4.3, one can comprehend that the NMPC was successful controlling the plant to maximize the profit while taking care of all constraints. Also the maximum produced gas constraint is active after around 2 minutes which shows that the answer for profit of the plant is optimal. The gas lift trajectory is oscillatory because the NMPC tries to maximize the profit of the plant while it has constraints on the way and the only mean to do these works is the gas lift. In other words the NMPC should maximize ω_{po} while maintaining the restriction on ω_{pg}. Considering the presented subplots for the produced oil in the system (subplots in the middle), one can say that because of the different productivity indices for three wells, the NMPC prefers to prioritize the attention on the wells based on the PI factor since the GOR in all wells is assumed to be equal. In other words, well number 3 which has the highest PI factor (3.2 KG/bar/s) always has the highest share from gas lift and it can be seen from the produced oil plot that the well number three made the highest amount of oil among the others. Well number 2 and 1 come after well number 3 respectively. Also, for the behavior of the total produced oil plot, one can argue that it is obvious that the total oil production drops after a while but the profit which has been made through making that jump is higher than the

case that optimizer does not make the overshoot and keep the steady state value of the total oil production at somewhere higher than the presented case. In other words, it is worth it to have a lower steady state value but have a considerable amount of produced oil in the beginning. It is also advantageous for total produced gas limitation since in the beginning the well network can accept having gas inside itself without having the total produced gas constraint active but passing the the time, the capacity for having gas inside the system reduces and it would be better to have less gas inside the wells.

4.4 Effect of Parameter GOR on Production Rate

In this section and the next section (4.5), the effect of changing the two most important parameters among the system characteristics which are *Gas Oil Ratio (GOR)* and productivity index *PI* is considered. In this section the parameter GOR has been changed to see the effect of that change on the production rate and consequently on the profit made in the plant. To have the result more visible and change only one factor at a time, the GOR parameter is assumed to be equal for all three wells during the examination and the set of GOR parameters is reduced by 50% to see the effect on the oil and gas production as well as the usage of gas lift for each case. To achieve that goal, the simulation is done twice, first with $GOR = [0.1, 0.1, 0.1]$ and results were recorded. The second time, the same test was done but with $GOR = [0.05, 0.05, 0.05]$ which was 50% lower than the first case. The results are shown in Figure 4.4 where the three subplots are presenting the results of two experiments by showing the gas lift usage, total oil produced and total gas produced respectively.

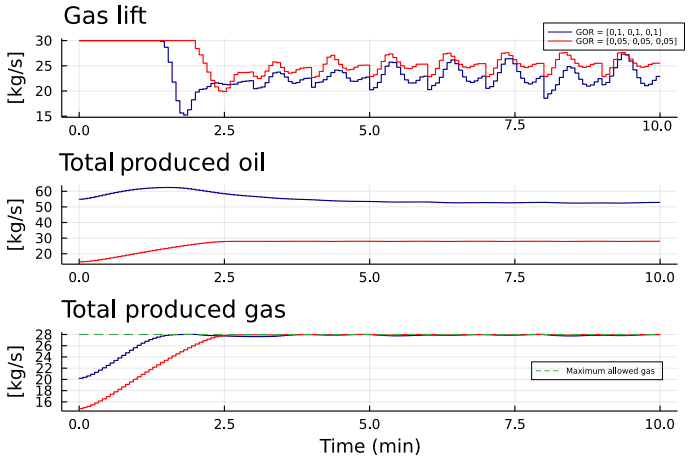


Figure 4.4: Representation of the effect of changing the GOR factor on the production rate. GOR has the value of 0.1 for all wells in blue line while it takes the value of 0.05 in red lines

Having Figure 4.4in mind, One can conclude that reducing the GOR would

not be beneficial for the system since it reduces the pressure difference between the reservoir and bottom hole of the well. As it can be seen from the total produced oil plot, production rate for the case where GOR is 0.05 is considerably less than the case with GOR 0.1 for all wells. It is also visible that having less gas to oil ratio would result in later activation of the total produced gas constraint which is true because the amount of gas present in the system in the first case is much less than the second case. Therefore, it would take longer for the system to reach the total gas limitation.

4.5 Effect of Parameter PI on Production Rate

The productivity index is the other important factor in the oil production plant which determines the possible amount of oil produced from a well compared to another one. In this section, changing this factor in order to capture the effect of it on the production rate is considered. As it was mentioned in the previous section, it has been tried to keep everything but the wanted factor the same before and after changing the factor in order to see its outcome more clearly. Figure 4.5 shows the experiment results where two cases were experimented: one with the nominal PI values $PI = [1.2, 2.2, 3.2]$ and the second one with 5% deduction in the values ($PI = [1.14, 2.09, 3.04]$). The results can be shown as what is presented in Figure 4.5:

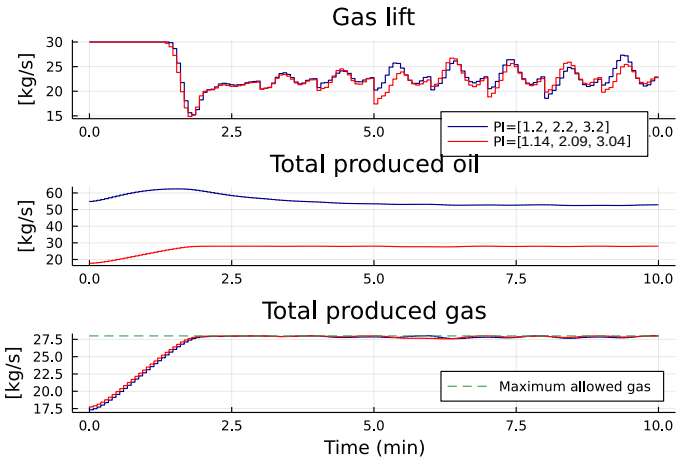


Figure 4.5: Illustration of the effect of changing the PI factor on the production rate. The well number 3 has the highest PI factor among the other wells which leads to have the greatest share in the total oil production

As it is obvious from the presented figure, less productivity index would results in less production rate. The steady state value for deducted case is around 30 [Kg/s] while the nominal case have the steady state value of around 50 [Kg/s]. Another way to examine this effect is to check the effect of having different productivity indices for wells in the network. As it was mentioned

before, having the higher productivity factor would result in producing more oil as it can be seen from Figure ... where three wells with nominal PI values are shown after a closed loot experiment.

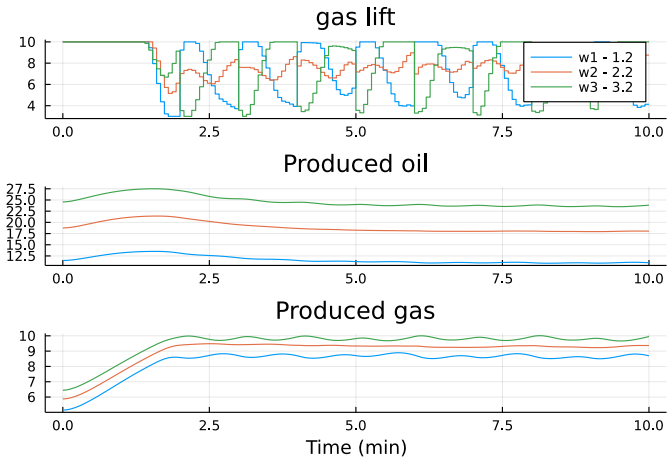


Figure 4.6: Illustration of the effect of the parameter PI on the production rate. Blue lines are the nominal values while the red ones have been reduced by 5% compared to the nominal case

4.6 Effect of Changing the Maximum Gas Constraint on controller behavior

In this section, testing the ability of the controller in facing constraint changing is considered. As an experiment, the plant was run for 15 minutes with changing the maximum allowed produced gas limitation in between. The nominal constraint for total produced gas was 28 [KG/s] and it was reduced by 5% to capture the behavior of the NMPC to move the plant to somewhere with less allowed produced gas. The results for this experiment is shown in Figure 4.7 where the data for both separated wells and total production is depicted.

As it can be captured from the figure 4.7, the NMPC was successful in controlling the maximum produced gas from the system while it made the best possible profit out of the system.

4.7 Possible Future Work

As it was also mentioned in the beginning of this thesis, all the uncertainties on parameters are neglected in this work and it is assumed that the uncertain parameters such as GOR would take their expected values. While in the actual plant it might not possible all the time. Because of that, the writer of this thesis has been doing an extra work as a continuation to this work in order to change

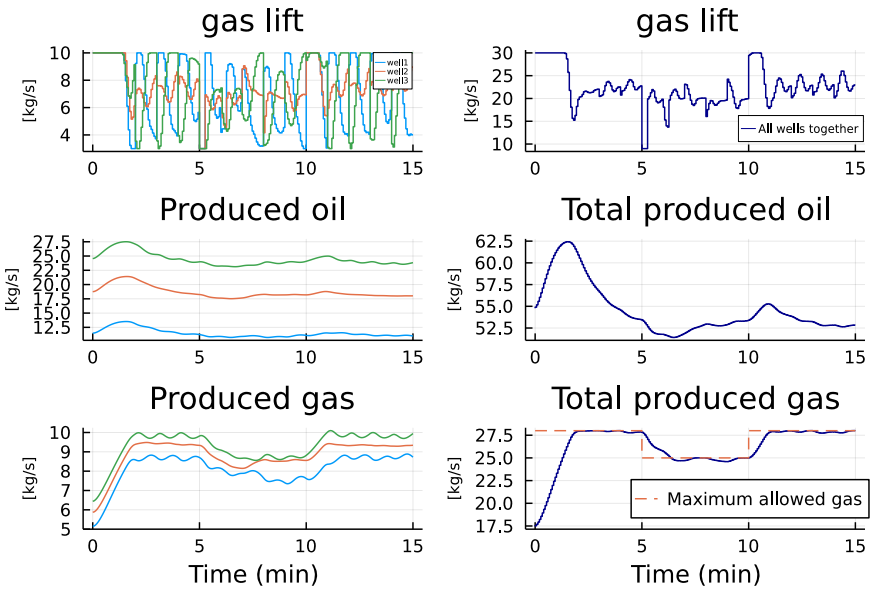


Figure 4.7: Illustration of the effect of Changing the maximum allowed produced gas limitation on the controller behavior

the NMPC to a Multistage Scenario-based Economic NMPC which can take all the possibilities for uncertain parameter(s) and create the scenario tree for the case. Completing that code, it can be attached to this plant to include the uncertainties as well. Moreover, the writer have the thought in mind to also apply decomposition algorithms on the Multistage Scenario-based Economic NMPC in order to address the probable long computation time for solving the problem. The code for multi stage scenario based NMPC is also attached to the appendices section. Note that the code is under construction and it might have several problems solving the oil production network problem.

Chapter 5

Conclusion

In this thesis, optimization a gas lifted oil production network is considered. The model has been defined as as DAE system which is imported into the julia programming language to simulate the actual model. Also the optimal control problem for the plant is created in order to maximize the profit of the plant. In other words, creating an economic model predictive control for the oil production network is considered. The optimal control problem is descretized and translated in a way that it is understandable for numerical solver in julia. To achieve that goal, the fourth order collocation method is taken into consideration to translate the dynamics of the system for julia in a way that a numerical method can solve that. In other words, the optimal control problem has been changed to a non-linear programming and fed into julia so that the numerical solver such as *IPopt* can handle it. Furthermore, the model for the plant is coded in julia and transformed to an ordinary differential system of equations for simplicity. The model is solved using the *BS3()* solver.The approach to simulate the NMPC procedure can be mentioned as below:

The initial values for the problem are given so that the steady state values can be found using the model integrator. Then the optimizer can solve an optimization problem over a prediction horizon that maximizes the profit of the plant as well as taking care of all constraints in the system. Afterwards, the modelled plant will take the first part of the optimal control sequence and solve the ODE system to find the states for the next time steps. The states for the next time step is fed into the optimizer again and the procedure will be repeated until it reaches the end point which is the simulation time.

In order to have the model tested for open loop and NMPC work, it has been used in different ways to capture different aspects of the plant such as changing important factors in the plant and recording the effect of that on the production rate. Or testing the NMPC in case of changing a constraint for instance maximum allowed produced gas. The results are presented and discussed in detail in the respective chapters.

All in all, the simulation shows that the model is a good approximation of the actual plant while the NMPC does not have any set point and it only

maximizes the profit of the plant. Also using the julia programming language made the computation of the results faster and it made the problem easier to be written and coded considering amazing packages pre-loaded in the programming language. The studies in this chapter led the writer to consider continuing the work after the master program as an extra work which is to include uncertainties to the problem and decomposition algorithm for addressing the computation time since the size of the problem is considerably large.

Bibliography

[1] A. Plucenio, D. Pagano, E. Camponogara, A. Traple and A. Teixeira, 'Gas-lift optimization and control with nonlinear mpc,' *IFAC Proceedings Volumes*, vol. 42, no. 11, pp. 904–909, 2009, 7th IFAC Symposium on Advanced Control of Chemical Processes, ISSN: 1474-6670. DOI: https://doi.org/10.3182/20090712-4-TR-2008.00148. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667015303918.

[2] G. O. Eikrem, O. M. Aamo and B. A. Foss, 'On Instability in Gas Lift Wells and Schemes for Stabilization by Automatic Control,' *SPE Production & Operations*, vol. 23, no. 02, pp. 268–279, May 2008, ISSN: 1930-1855. DOI: 10.2118/101502-PA. eprint: https://onepetro.org/PO/article-pdf/23/02/268/2108429/spe-101502-pa.pdf. [Online]. Available: https://doi.org/10.2118/101502-PA.

[3] A. Satter, G. Iqbal and J. Buchwalter, *Practical enhanced reservoir engineering: assisted with simulation software*. PennWell Books, LLC, 2008.

[4] M. L. Hammadih, K. Al-Hosani and I. Boiko, 'Soft sensing in deep wells within artificial gas lift technology,' Nov. 2015. DOI: 10.2118/177731-MS.

[5] D. Krishnamoorthy, J. Ryu and S. Skogestad, 'A dynamic extremum seeking scheme applied to gas lift optimization,' *IFAC-PapersOnLine*, vol. 52, no. 1, pp. 802–807, 2019, 12th IFAC Symposium on Dynamics and Control of Process Systems, including Biosystems DYCOPS 2019, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2019.06.160. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896319302472.

[6] J. Nocedal and S. Wright, *Numerical optimization*. Springer Science & Business Media, 2006.

[7] B. Foss and T. A. N. Heirung, 'Merging optimization and control,' *Lecture Notes*, Mar. 2016. [Online]. Available: https://www.researchgate.net/publication/301685613_Merging_Optimization_and_Control.

[8] M. Diehl and S. Gros, 'Numerical optimal control,' *Optimization in Engineering Center (OPTEC)*, 2011.

[9] J. Jäschke, 'Dynamic optimization for mpc,' *Lecture Notes*, Sep. 2020.

[10] M. F. Dlima, 'Nonlinear model predictive control of gravity separators,' M.S. thesis, NTNU, 2017.

[11] M. Osborne, 'On shooting methods for boundary value problems,' *Journal of Mathematical Analysis and Applications*, vol. 27, no. 2, pp. 417–433, 1969, ISSN: 0022-247X. DOI: https://doi.org/10.1016/0022-247X(69)90059-6. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0022247X69900596.

[12] C. Kirches, 'Fast numerical methods for mixed-integer nonlinear model-predictive control,' Ph.D. dissertation, Heidelberg Graduate School of Mathematical and Computational Methods for Sciences, 2010.

[13] L. T. Biegler, *Nonlinear Programming*. Society for Industrial and Applied Mathematics, 2010. DOI: 10.1137/1.9780898719383. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9780898719383. [Online]. Available: https://epubs.siam.org/doi/abs/10.1137/1.9780898719383.

[14] S. Skogestad, 'Plantwide control: The search for the self-optimizing control structure,' *Journal of Process Control*, vol. 10, no. 5, pp. 487–507, 2000, ISSN: 0959-1524. DOI: https://doi.org/10.1016/S0959-1524(00)00023-8. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959152400000238.

[15] D. Krishnamoorthy, B. Foss and S. Skogestad, 'Real-time optimization under uncertainty applied to a gas lifted well network,' *Processes*, vol. 4, no. 4, p. 52, 2016.

[16] D. Krishnamoorthy, K. Fjalestad and S. Skogestad, 'Optimal operation of oil and gas production using simple feedback control structures,' *Control Engineering Practice*, vol. 91, p. 104 107, 2019.

[17] S. H. Yahia, 'Real time optimization and nonlinear model predictive control of a gas lifted oil network,' M.S. thesis, NTNU, 2018.

[18] A. Zamani Meighani, *Developing an mpc package with julia programming language, specialization project,ntnu, 2020.*

[19] *Julia documentation user manual*, https://docs.julialang.org/en/v1/, Accessed: 2022-01-01.

[20] *JuMP documentation user manual*, https://jump.dev/JuMP.jl/stable/, Accessed: 2022-01-01.

[21] *Ipopt documentation user manual*, https://github.com/jump-dev/Ipopt.jl, Accessed: 2022-01-01.

Appendix A

Additional Material

Parameter	Units	Well 1	Well 2	Well 3
L_w	[m]	1500	1500	1500
H_w	[m]	1000	1000	1000
D_w	[m]	0.121	0.121	0.121
L_{bh}	[m]	500	500	500
H_{bh}	[m]	500	500	500
D_{bh}	[m]	0.121	0.121	0.121
L_a	[m]	1500	1500	1500
H_a	[m]	1000	1000	1000
D_a	[m]	0.189	0.189	0.189
ρ_0	[Kg m^-3]	800	800	800
C_{iv}	[m^2]	0.1E-3	0.1E-3	0.1E-3
C_{pc}	[m^2]	2E-3	2E-3	2E-3
P_r	[bar]	150	150	150
PI	[K/s/bar]	1.2	2.2	3.2
T_a	[C]	28	28	28
T_w	[C]	32	32	32
GOR	[Kg/Kg]	0.1	0.1	0.1

Table A.1: Parameter used in the system. Based on the specific experiments, some of the might have been used with different values which in that case, it is explained in the respective section.

A.1 Julia Codes

For the last version of the codes, please see:
$https://github.com/Amirrezz94/Gas_Lift_Final$

A Plant.jl

```
include("Parameters.jl")

using DifferentialEquations
using Plots
using DataInterpolations

function Plant(Nw, U_opt, States, t_min)
    if Nw == 1
        par = WellPar1()
        @unpack_WellPar1 par
    elseif Nw == 2
        par = WellPar2()
        @unpack_WellPar2 par
    elseif Nw == 3
        par = WellPar3diff()
        @unpack_WellPar3diff par
    else
        println("Nw must be either 1 or 2 or 3!")
        #More data can be added in Parameters.jl
    end

    ##function
    function Gas_Lift_Model(dx,x,p,t)
        ## mass flow rates
        _gl = p(t)
        m_ga = x[1:Nw, 1] # mass of gas in annulus
        m_gt = x[1:Nw, 2] # mass of gas in tubing
        m_ot = x[1:Nw, 3] # mass of oil in tubing
        ## Algabraic Equations
        P_a = 1e-5*(((R.*T_a./(V_a.*Mw) + 9.81.*H_a./V_a).*m_ga.*1e3) +
        ↪  (Mw./(R.*T_a).*((R.*T_a./(V_a.*Mw) + 9.81.*H_a./V_a).*m_ga.*1e3)).*9.81.*H_a)
        ↪  # annulus pressure
        _a = 1e-2*(Mw./(R.*T_a).*P_a*1e5)  # gas, in annulus
        P_wh = 1e-5*(((R.*T_w./Mw).*(m_gt*1e3./(L_w.*A_w + L_bh.*A_bh .- m_ot.*1e3./))) -
        ↪  ((m_gt*1e3+m_ot*1e3 ) ./ (L_w.*A_w)).*9.81.*H_w./2) # wellhead pressure
        rho_m = 1e-2*(((m_gt*1e3 + m_ot*1e3).*P_wh*1e5.*Mw.*) ./ (m_ot*1e3.*P_wh*1e5.*Mw +
        ↪  .*R.*T_w.*m_gt*1e3))  # mixture, in tubing
        _pc = C_pc.*sqrt.(rho_m*1e2.*(P_wh*1e5 .- P_m*1e5)) # total flow through choke
        P_wi = 1e-5*((P_wh*1e5 + 9.81./(A_w .* L_w) .* (m_ot*1e3+m_gt*1e3 .-  .* L_bh .*
        ↪  A_bh) .* H_w + 128*mu_oil .* L_w .* _pc ./(3.141*D_w .^ 4 .*((m_gt*1e3 +
        ↪  m_ot*1e3) .* P_wh * 1e5 .* Mw .* ) ./ (m_ot*1e3 .* P_wh*1e5.*Mw +  .* R .* T_w
        ↪  .* m_gt * 1e3))))
        _iv = C_iv .*sqrt.(_a*1e2.*(P_a*1e5 - P_wi*1e5))
        _pg = (m_gt*1e3./(m_gt*1e3+m_ot*1e3)).*_pc # produced gas rate
        _po = (m_ot*1e3./(m_gt.*1e3+m_ot.*1e3)).*_pc # produced oil rate
        P_bh = 1e-5*(P_wi*1e5 .+ *9.81.*H_bh + 128*mu_oil.*L_bh.*_po./(3.14 * D_bh .^4 .*
        ↪  ))
        _ro = (PI)*1e-6.*(Press_r*1e5 .- P_bh*1e5)
        _rg = 1e1*GOR.*_ro
        ## Differential Equations
        dx[1:Nw, 1] = (_gl - _iv)*1e-3
        dx[1:Nw, 2] = (_iv + _rg*1e-1 - _pg)*1e-3
        dx[1:Nw, 3] = (_ro - _po)*1e-3
    end
```

```
47
48      tspan = (0.0, t_min*60)
49      t_points = collect(tspan[1]:4:tspan[2])
50
51      gas_lift_1 = ConstantInterpolation(U_opt[1,:],t_points)
52      gas_lift_2 = ConstantInterpolation(U_opt[2,:],t_points)
53      gas_lift_3 = ConstantInterpolation(U_opt[3,:],t_points)
54
55          p_fun(t) = [gas_lift_1(t), gas_lift_2(t), gas_lift_3(t)]
56
57      # I assumed U_opt is [1:3,1:16]
58      #Problem Loading
59      prob = ODEProblem(Gas_Lift_Model, States[:x0], tspan, p_fun)
60
61      #Solve
62      sol = DifferentialEquations.solve(prob, BS3(),saveat=4)
63      # sol = DifferentialEquations.solve(prob,saveat=4)
64          #storing the trajectories
65          temp_1 = Array(sol)[1,:,:]
66      temp_2 = Array(sol)[2,:,:]
67      temp_3 = Array(sol)[3,:,:]
68      m_ga_profile = [temp_1[1,:]';temp_2[1,:]';temp_3[1,:]']
69      m_gt_profile = [temp_1[2,:]';temp_2[2,:]';temp_3[2,:]']
70      m_ot_profile = [temp_1[3,:]';temp_2[3,:]';temp_3[3,:]']
71
72
73      U_profile = zeros(3,length(t_points))
74      for i =1:length(t_points)
75          U_profile[:,i] = p_fun(t_points[i])
76      end
77
78  ##=================================================
79      # m_ga0 = sol.u[end][:, 1]
80      # m_gt0 = sol.u[end][:, 2]
81      # m_ot0 = sol.u[end][:, 3]
82          state0 = [m_ga_profile[:,end] m_gt_profile[:,end] m_ot_profile[:,end]]
83      # state0 = [temp_1[:,end] temp_2[:,end] temp_3[:,end]]
84      ##
85      # dxi =zeros(Nw,3)
86      P_a = 1e-5*(((R.*T_a./(V_a.*Mw) + 9.81.*H_a./V_a).*m_ga_profile.*1e3) +
        ↪  (Mw./(R.*T_a).*((R.*T_a./(V_a.*Mw) +
        ↪  9.81.*H_a./V_a).*m_ga_profile.*1e3)).*9.81.*H_a) # annulus pressure
87      _a = 1e-2*(Mw./(R.*T_a).*P_a*1e5)  # gas, in annulus
88      P_wh = 1e-5*(((R.*T_w./Mw).*(m_gt_profile*1e3./(L_w.*A_w + L_bh.*A_bh .-
        ↪  m_ot_profile.*1e3./))) - ((m_gt_profile*1e3+m_ot_profile*1e3 ) ./
        ↪  (L_w.*A_w)).*9.81.*H_w./2) # wellhead pressure
89      rho_m = 1e-2*(((m_gt_profile*1e3 + m_ot_profile*1e3).*P_wh*1e5.*Mw.*) ./
        ↪  (m_ot_profile*1e3.*P_wh*1e5.*Mw + .*R.*T_w.*m_gt_profile*1e3))  # mixture, in
        ↪  tubing
90      _pc = C_pc.*sqrt.(rho_m*1e2.*(P_wh*1e5 .- P_m*1e5)) # total flow through choke
91      P_wi = 1e-5*((P_wh*1e5 + 9.81./(A_w .* L_w) .* (m_ot_profile*1e3+m_gt_profile*1e3 .-
        ↪  .* L_bh .* A_bh) .* H_w + 128*mu_oil .* L_w .* _pc ./(3.141*D_w .^ 4
        ↪  .*((m_gt_profile*1e3 + m_ot_profile*1e3) .* P_wh * 1e5 .* Mw .* ) ./
        ↪  (m_ot_profile*1e3 .* P_wh*1e5.*Mw +  .* R .* T_w .* m_gt_profile * 1e3))))
92      _iv = C_iv .*sqrt.(_a*1e2.*(P_a*1e5 - P_wi*1e5))
93      _pg = (m_gt_profile*1e3./(m_gt_profile*1e3+m_ot_profile*1e3)).*_pc # produced gas rate
94      _po = (m_ot_profile*1e3./(m_gt_profile.*1e3+m_ot_profile.*1e3)).*_pc # produced oil
        ↪  rate
95      P_bh = 1e-5*(P_wi*1e5 .+ *9.81.*H_bh + 128*mu_oil.*L_bh.*_po./(3.14 * D_bh .^4 .* ))
96      _ro = (PI)*1e-6.*(Press_r*1e5 .- P_bh*1e5)
97      _rg = 1e1*GOR.*_ro
98      dx1 = (U_profile - _iv)*1e-3
99      dx2 = (_iv + _rg*1e-1 - _pg)*1e-3
```

```
        U_opt, _po, _pg, obj = Plant_optimizer(U_opt, t_min_o, t_min_p, Nw, Initial, p, _gl, , false);
        Inits = Plant(Nw, U_opt, Initial, t_min_p);

        U_opt_hist   = U_opt;
        _po_profile = Inits[:_po0];
        _pg_profile = Inits[:_pg0];



        for k in 2:5
                U_opt, _po, _pg,obj  = Plant_optimizer(U_opt, t_min_o, t_min_p, Nw, Inits, p, _gl, , false);
                U_opt_hist = hcat(U_opt_hist, U_opt[:, 2:end]);
                Inits= Plant(Nw, U_opt, Inits, t_min_p);

                _po_profile = hcat(_po_profile, Inits[:_po0][:, 2:end]);
                _pg_profile = hcat(_pg_profile, Inits[:_pg0][:, 2:end]);
        end
        for k in 6:10
                U_opt, _po, _pg,obj  = Plant_optimizerr(U_opt, t_min_o, t_min_p, Nw, Inits, p, _gl, , false);
                U_opt_hist = hcat(U_opt_hist, U_opt[:, 2:end]);
                Inits= Plant(Nw, U_opt, Inits, t_min_p);

                _po_profile = hcat(_po_profile, Inits[:_po0][:, 2:end]);
                _pg_profile = hcat(_pg_profile, Inits[:_pg0][:, 2:end]);
        end
        for k in 11:15
                U_opt, _po, _pg,obj  = Plant_optimizer(U_opt, t_min_o, t_min_p, Nw, Inits, p, _gl, , false);
                U_opt_hist = hcat(U_opt_hist, U_opt[:, 2:end]);
                Inits= Plant(Nw, U_opt, Inits, t_min_p);

                _po_profile = hcat(_po_profile, Inits[:_po0][:, 2:end]);
                _pg_profile = hcat(_pg_profile, Inits[:_pg0][:, 2:end]);
        end


println("Simulation has ended!")

## plotting

plotlyjs()
gr()
t_points = collect(0:4:t_min_s*60)
Max_Gas = vec(hcat(28*ones(1, 15*5 + 1), 25*ones(1, 15 * 5 ), 28*ones(1, 15 * 5 )))
p1=plot(t_points/60,U_opt_hist', linetype=:steppre, title="gas lift",label=["well1" "well2" "well3"])
p_po=plot(t_points/60,_po_profile', title="Produced oil",label=false)
p_pg=plot(t_points/60,_pg_profile', title="Produced gas",xlabel = "Time (min)",label=false)
fig1 = plot(p1,p_po,p_pg , layout = (3, 1))
##
t_points = collect(0:4:t_min_s*60)
p1=plot(t_points/60,    sum(U_opt_hist[i,:] for i in 1:Nw) , linetype=:steppre,title="gas lift",label="All wells together",color = :bluesreds, legend = (0.4, 0.3))
p_po=plot(t_points/60,  sum(_po_profile[i,:] for i in 1:Nw) , linetype=:steppre,title="Total produced oil",label=false,color = :bluesreds)
p_pg=plot(t_points/60,  sum(_pg_profile[i,:] for i in 1:Nw) , linetype=:steppre,xlabel = "Time (min)",label=false,color = :bluesreds)
p_pg=plot!(t_points/60,  Max_Gas , linetype=:steppre,title="Total produced gas",xlabel = "Time (min)",label=" Maximum allowed gas", linestyle=:dash, legend = (0.4, 0.3))
```

```
fig2 = plot(p1,p_po,p_pg , layout = (3, 1))

fig3 =  plot(fig1, fig2 , layout = (1, 2), yaxis="[kg/s]", yguidefontsize=7)

savefig(fig3, "Figures//11Jan//Results_part4.pdf")



#p1=plot(x,[ua u],xlabel = "x", ylabel = "u", color = [:deepskyblue :red],  label =
  ↪  ["Exact" "Numerical"], lw = 2,linealpha=[1.0 .2],marker = ( [:none :o],0.75,
  ↪  Plots.stroke(:black)),markersize=2, legend=(-35.0, 1.0),framestyle = :box, grid=false)
#julia>
  ↪  plot(rand(40),xtickfontsize=18,ytickfontsize=18,xlabel="wavelength",xguidefontsize=18,yscale=:log10,ylabe
  ↪  here
##Save the Figures
using Dates
a = today()
savefig(fig3, "Figures//11Jan//abcd.pdf")
## Profit Computing

profit = _p * sum(_po_profile) - _gl * sum(U_opt_hist)
##Data recording center
U_opt_gmaxr = U_opt_hist
_po_gmaxr = _po_profile
_pg_gmaxr = _pg_profile
#--------------------------------
U_opt_gmaxp5 = U_opt_hist;
 _po_gmaxp5 = _po_profile;
 _pg_gmaxp5 = _pg_profile;
#--------------------------------
U_opt_gmaxm5 = U_opt_hist;
 _po_gmaxm5 = _po_profile;
 _pg_gmaxm5 = _pg_profile;
##



p1=plot(t_points/60,U_opt_gmaxr', linetype=:steppre, title="gas lift",label=["w1 - 1.2"
  ↪  "w2 - 2.2" "w3 - 3.2"])
p_po=plot(t_points/60,_po_gmaxr', title="Produced oil",label=false)
p_pg=plot(t_points/60,_pg_gmaxr', title="Produced gas",xlabel = "Time (min)",label=false)
fig1 = plot(p1,p_po,p_pg , layout = (3, 1))









t_points = collect(0:4:t_min_s*60)

p1=plot(t_points/60,    sum(U_opt_gmaxr[i,:] for i in 1:Nw) , linetype=:steppre,title="Gas
  ↪  lift",label="PI=[1.2, 2.2, 3.2]",color = :bluesreds, labelsize=50)
```

```
p1=plot!(t_points/60,    sum(U_opt_gmaxp5[i,:] for i in 1:Nw) ,
    linetype=:steppre,title="Gas lift",label="PI=[1.26, 2.31, 3.36]",color = :red,
    labelsize=50)
# p1=plot!(t_points/60,    sum(U_opt_gmaxp5[i,:] for i in 1:Nw) ,
    linetype=:steppre,title="gas lift",label="PI=[1.14, 2.09, 3.04]",color = :green)

p_po=plot(t_points/60,  sum(_po_gmaxr[i,:] for i in 1:Nw) , linetype=:steppre,title="Total
    produced oil",label=false,color = :bluesreds)
p_po=plot!(t_points/60,  sum(_pg_gmaxp5[i,:] for i in 1:Nw) ,
    linetype=:steppre,title="Total produced oil",label=false,color = :red)
# p_po=plot!(t_points/60,  sum(_pg_gmaxp5[i,:] for i in 1:Nw) ,
    linetype=:steppre,title="Total produced oil",label=false,color = :green)

p_pg=plot(t_points/60,  sum(_pg_profile[i,:] for i in 1:Nw) , linetype=:steppre,xlabel =
    "Time (min)",label=false,color = :bluesreds)
p_pg=plot!(t_points/60,  sum(_pg_gmaxp5[i,:] for i in 1:Nw) , linetype=:steppre,xlabel =
    "Time (min)",label=false,color = :red)
# p_pg=plot!(t_points/60,  sum(_pg_gmaxp5[i,:] for i in 1:Nw) , linetype=:steppre,xlabel =
    "Time (min)",label=false,color = :green)

p_pg=plot!(t_points/60,  28 * ones(length(t_points)) , linetype=:steppre,title="Total
    produced gas",xlabel = "Time (min)",label=" Maximum allowed gas", linestyle=:dash)
fig4 = plot(p1,p_po,p_pg , layout = (3, 1), legend = (0.7, -.1), yaxis="[kg/s]")


savefig(fig4, "Figures//11Jan//Results_part3.pdf")


profitr = _p * sum(_po_gmaxr) - _gl * sum(U_opt_gmaxr)
profitp = _p * sum(_pg_gmaxm5) - _gl * sum(U_opt_gmaxm5)
```

C Optimizer.jl

```
using JuMP
using Ipopt
using Plots
##Supplementary Files
include("CollMat.jl")
include("Parameters.jl")
include("InitiStates.jl")
#choose backend for plots
# plotlyjs()

##Define the optimizer Function
function Plant_optimizer(u0, t_min, t_min_inp, Nw, Inits, _p, _gl, , Plotting)
    #Required Parameters
    Nx = 3;
    Nz = 12;
    Nu = 1;
    NCP = 4
    # NFE = round(Int, t_min/NCP)
    NFE = round(Int, t_min * 60/NCP)
    # Nw  =  3;


    # u0=[4.0; 4.0; 4.0]

    # Inits = Initials()
    # @unpack_Initials Inits
```

```
27
28      m_ga0    = Inits[:m_ga_pro]
29      m_gt0    = Inits[:m_gt_pro]
30      m_ot0    = Inits[:m_ot_pro]
31      P_a0     = Inits[:P_a0]
32      _a0     = Inits[:_a0]
33      P_wh0    = Inits[:P_wh0]
34      _m0     = Inits[:_m0]
35      _pc0    = Inits[:_pc0]
36      P_wi0    = Inits[:P_wi0]
37      _iv0    = Inits[:_iv0]
38      _pg0    = Inits[:_pg0]
39      _po0    = Inits[:_po0]
40      P_bh0    = Inits[:P_bh0]
41      _ro0    = Inits[:_ro0]
42      _rg0    = Inits[:_rg0]
43      dx0      = Inits[:dx0]
44      x0       = Inits[:x0]
45
46      M = Collocation_Matrix(NCP)
47      dt = 4;
48      #  p = 1;
49      # _gl = 0.01;
50      #  = 0;
51
52      # Parameter Loading
53      if Nw == 1
54          par = WellPar1()
55          @unpack_WellPar1 par
56      elseif Nw == 2
57          par = WellPar2()
58          @unpack_WellPar2 par
59      elseif Nw == 3
60          # par = WellPar3()
61          # @unpack_WellPar3 par
62          par = WellPar3diff()
63          @unpack_WellPar3diff par
64      else
65          println("Nw must be either 1 or 2 or 3!")
66          #More data can be added in Parameters.jl
67      end
68
69      ##Model Defining
70      m1 = Model(Ipopt.Optimizer)
71      #Defining the Variables=============================================================
72      @variable(m1,  x[1:Nw, 1:Nx, 1:NFE, 1:NCP+1])
73      @variable(m1, dx[1:Nw, 1:Nx, 1:NFE, 1:NCP  ])
74      @variable(m1,  z[1:Nw, 1:Nz, 1:NFE, 1:NCP  ])
75      @variable(m1,  u[1:Nw, 1:Nu, 1:NFE         ])
76      #Bounds and Starting Points=========================================================
77      for nw in 1:Nw, nx in 1:Nx, nz in 1:Nz, nfe in 1:NFE, ncp in 1:NCP, nu in 1:Nu
78          #Bounds
79          set_lower_bound(x[nw, nx, nfe, ncp], 0.01)
80          set_upper_bound(x[nw, nx , nfe, ncp], 1e7)
81          set_lower_bound(z[nw, nz, nfe, ncp], 0)
82          set_upper_bound(z[nw, nz , nfe, ncp], 1e7)
83          set_lower_bound(u[nw, nu, nfe], 3)
84          set_upper_bound(u[nw, nu, nfe], 10)
85
86      end
87      #Start Values
88      for nw in 1:Nw, nfe in 1:t_min_inp*15, ncp in 1:NCP
89          set_start_value(x[nw, 1, nfe, ncp],      m_ga0[nw, nfe])
```

```
        set_start_value(x[nw, 2, nfe, ncp],      m_gt0[nw, nfe])
        set_start_value(x[nw, 3, nfe, ncp],      m_ot0[nw, nfe])

        set_start_value(u[nw, 1, nfe],          u0[nw, nfe])

        set_start_value(z[nw, 1, nfe, ncp],     P_a0[nw, nfe])
        set_start_value(z[nw, 2, nfe, ncp],     _a0[nw, nfe])
        set_start_value(z[nw, 3, nfe, ncp],     _m0[nw, nfe])
        set_start_value(z[nw, 4, nfe, ncp],     P_wh0[nw, nfe])
        set_start_value(z[nw, 5, nfe, ncp],     P_wi0[nw, nfe])
        set_start_value(z[nw, 6, nfe, ncp],     P_bh0[nw, nfe])
        set_start_value(z[nw, 7, nfe, ncp],     _iv0[nw, nfe])
        set_start_value(z[nw, 8, nfe, ncp],     _pc0[nw, nfe])
        set_start_value(z[nw, 9, nfe, ncp],     _pg0[nw, nfe])
        set_start_value(z[nw, 10, nfe, ncp],    _po0[nw, nfe])
        set_start_value(z[nw, 11, nfe, ncp],    _ro0[nw, nfe])
        set_start_value(z[nw, 12, nfe, ncp],    _rg0[nw, nfe])
    end
    for nw in 1:Nw, nfe in t_min_inp*15 + 1:NFE, ncp in 1:NCP
        set_start_value(x[nw, 1, nfe, ncp],      m_ga0[nw, end])
        set_start_value(x[nw, 2, nfe, ncp],      m_gt0[nw, end])
        set_start_value(x[nw, 3, nfe, ncp],      m_ot0[nw, end])

        set_start_value(u[nw, 1, nfe],          u0[nw, end])

        set_start_value(z[nw, 1, nfe, ncp],     P_a0[nw, end])
        set_start_value(z[nw, 2, nfe, ncp],     _a0[nw, end])
        set_start_value(z[nw, 3, nfe, ncp],     _m0[nw, end])
        set_start_value(z[nw, 4, nfe, ncp],     P_wh0[nw, end])
        set_start_value(z[nw, 5, nfe, ncp],     P_wi0[nw, end])
        set_start_value(z[nw, 6, nfe, ncp],     P_bh0[nw, end])
        set_start_value(z[nw, 7, nfe, ncp],     _iv0[nw, end])
        set_start_value(z[nw, 8, nfe, ncp],     _pc0[nw, end])
        set_start_value(z[nw, 9, nfe, ncp],     _pg0[nw, end])
        set_start_value(z[nw, 10, nfe, ncp],    _po0[nw, end])
        set_start_value(z[nw, 11, nfe, ncp],    _ro0[nw, end])
        set_start_value(z[nw, 12, nfe, ncp],    _rg0[nw, end])
    end


    #Expressions Variables=====================================================
    #(makes it easier to write DAE Equation) #?add variables and change indices
    @NLexpressions(m1, begin
        m_ga[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP+1],          x[nw, 1, nfe, ncp]
        m_gt[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP+1],          x[nw, 2, nfe, ncp]
        m_ot[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP+1],          x[nw, 3, nfe, ncp]

        dm_ga[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],          dx[nw, 1, nfe, ncp]
        dm_gt[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],          dx[nw, 2, nfe, ncp]
        dm_ot[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],          dx[nw, 3, nfe, ncp]

        P_a[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],           z[nw, 1, nfe, ncp]
        _a[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],           z[nw, 2, nfe, ncp]
        _m[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],           z[nw, 3, nfe, ncp]
        P_wh[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],          z[nw, 4, nfe, ncp]
        P_wi[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],          z[nw, 5, nfe, ncp]
        P_bh[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],          z[nw, 6, nfe, ncp]
        _iv[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],          z[nw, 7, nfe, ncp]
        _pc[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],          z[nw, 8, nfe, ncp]
        _pg[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],          z[nw, 9, nfe, ncp]
        _po[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],          z[nw, 10, nfe, ncp]
        _ro[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],          z[nw, 11, nfe, ncp]
```

```
153         _rg[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],           z[nw, 12, nfe, ncp]
154
155         _gl[nw in 1:Nw, nfe in 1:NFE],                          u[nw, 1, nfe]
156     end)
157     #Set the ODEs============================================================
158     @NLconstraints(m1, begin
159         ODE1[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],    dm_ga[nw, nfe, ncp]      ==
        ↪  (_gl[nw, nfe] - _iv[nw, nfe, ncp])*1e-3
160         ODE2[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],    dm_gt[nw, nfe, ncp]      ==
        ↪  (_iv[nw, nfe, ncp] - _pg[nw, nfe, ncp] + _rg[nw, nfe, ncp] * 1e-1)*1e-3
161         ODE3[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],    dm_ot[nw, nfe, ncp]      ==
        ↪  (_ro[nw, nfe, ncp] - _po[nw, nfe, ncp])*1e-3
162         #ODEn[nfe in 1:NFE, ncp in 1:NCP],   ...
163     end)
164     #Algebraic Equations=====================================================
165     @NLconstraints(m1, begin
166     #Defining Model Algebraic Equations in each line
167         Constr_Alg1[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],    P_a[nw, nfe, ncp]   ==
        ↪  1e-5*(((R[nw]*T_a[nw]/(V_a[nw]*Mw[nw]) + g[nw]*H_a[nw]/V_a[nw])*m_ga[nw, nfe,
        ↪  ncp]*1e3) + (Mw[nw]/(R[nw]*T_a[nw]))*((R[nw]*T_a[nw]/(V_a[nw]*Mw[nw]) +
        ↪  g[nw]*H_a[nw]/V_a[nw])*m_ga[nw, nfe, ncp]*1e3))*g[nw]*H_a[nw]) # annulus
        ↪  pressure
168         Constr_Alg2[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],    _a[nw, nfe, ncp]   ==
        ↪  1e-2*(Mw[nw]/(R[nw]*T_a[nw]))*P_a[nw, nfe, ncp]*1e5)  # gas, in annulus
169         Constr_Alg3[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],    P_wh[nw, nfe, ncp]  ==
        ↪  1e-5*(((R[nw]*T_w[nw]/Mw[nw])*(m_gt[nw, nfe, ncp]*1e3/(L_w[nw]*A_w[nw] +
        ↪  L_bh[nw]*A_bh[nw] - m_ot[nw, nfe, ncp]*1e3/[nw]))) - ((m_gt[nw, nfe,
        ↪  ncp]*1e3+m_ot[nw, nfe, ncp]*1e3 )/(L_w[nw]*A_w[nw]))*g[nw]*H_w[nw]/2) #
        ↪  wellhead pressure
170         Constr_Alg4[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],    _m[nw, nfe, ncp]   ==
        ↪  1e-2*(((m_gt[nw, nfe, ncp]*1e3 + m_ot[nw, nfe, ncp]*1e3)*P_wh[nw, nfe,
        ↪  ncp]*1e5*Mw[nw]*[nw])/(m_ot[nw, nfe, ncp]*1e3*P_wh[nw, nfe, ncp]*1e5*Mw[nw] +
        ↪  [nw]*R[nw]*T_w[nw]*m_gt[nw, nfe, ncp]*1e3))  # mixture, in tubing
171         Constr_Alg5[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],    _pc[nw, nfe, ncp]  ==
        ↪  C_pc[nw]*((_m[nw, nfe, ncp]*1e2*(P_wh[nw, nfe, ncp]*1e5 -
        ↪  P_m[nw]*1e5))^2)^(1/4) # total flow through choke
172         Constr_Alg6[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],    P_wi[nw, nfe, ncp]  ==
        ↪  1e-5*((P_wh[nw, nfe, ncp]*1e5 + g[nw]/(A_w[nw]*L_w[nw])*(m_ot[nw, nfe,
        ↪  ncp]*1e3+m_gt[nw, nfe, ncp]*1e3-[nw]*L_bh[nw]*A_bh[nw])*H_w[nw] +
        ↪  128*mu_oil[nw]*L_w[nw]*_pc[nw, nfe, ncp]/(3.141*D_w[nw]^4*((m_gt[nw, nfe,
        ↪  ncp]*1e3 + m_ot[nw, nfe, ncp]*1e3)*P_wh[nw, nfe,
        ↪  ncp]*1e5*Mw[nw]*[nw])/(m_ot[nw, nfe, ncp]*1e3*P_wh[nw, nfe, ncp]*1e5*Mw[nw] +
        ↪  [nw]*R[nw]*T_w[nw]*m_gt[nw, nfe, ncp]*1e3))))
173         Constr_Alg7[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],    _iv[nw, nfe, ncp]  ==
        ↪  C_iv[nw]*((_a[nw, nfe, ncp]*1e2*(P_a[nw, nfe, ncp]*1e5 - P_wi[nw, nfe,
        ↪  ncp]*1e5))^2)^(1/4)
174         Constr_Alg8[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],    _pg[nw, nfe, ncp]  ==
        ↪  (m_gt[nw, nfe, ncp]*1e3/(m_gt[nw, nfe, ncp]*1e3+m_ot[nw, nfe,
        ↪  ncp]*1e3))*_pc[nw, nfe, ncp] # produced gas rate
175         Constr_Alg9[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],    _po[nw, nfe, ncp]  ==
        ↪  (m_ot[nw, nfe, ncp]*1e3/(m_gt[nw, nfe, ncp]*1e3+m_ot[nw, nfe,
        ↪  ncp]*1e3))*_pc[nw, nfe, ncp] # produced oil rate
176         Constr_Alg10[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],   P_bh[nw, nfe, ncp]  ==
        ↪  1e-5*(P_wi[nw, nfe, ncp]*1e5 + [nw]*g[nw]*H_bh[nw] +
        ↪  128*mu_oil[nw]*L_bh[nw]*_po[nw, nfe, ncp]/(3.14*D_bh[nw]^4*[nw]))
177         Constr_Alg11[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],   _ro[nw, nfe, ncp]  ==
        ↪  (PI[nw])*1e-6*(Press_r[nw]*1e5 - P_bh[nw, nfe, ncp]*1e5)
178         Constr_Alg12[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],   _rg[nw, nfe, ncp]  ==
        ↪  1e1*GOR[nw]*_ro[nw, nfe, ncp]
179         #Constr_Alg999[nfe=1:NFE, ncp=1:NCP], alg[999,nfe,ncp] ==
180         Constr_InEq1[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],   (P_wh[nw, nfe, ncp] -
        ↪  P_m[nw])*1e5 >= 0
```

```
181         Constr_InEq2[nw in 1:Nw, nfe in 1:NFE, ncp in 1:NCP],   (P_a[nw, nfe, ncp] -
            ↪  P_wi[nw, nfe, ncp])*1e5 >= 0
182     end)
183
184     #Collcation!============================================================
185     @NLconstraints(m1, begin
186         Coll_Eq_Diff[nw in 1:Nw,     nx in 1:Nx,  nfe in 1:NFE,    ncp in 1:NCP],
            ↪  x[nw, nx, nfe, ncp+1] == x[nw, nx, nfe, 1] + sum(M[ncp, i] * dt * dx[nw, nx,
            ↪  nfe, i] for i in 1:NCP)
187         Cont_Eq_First[nw in 1:Nw,    nx in 1:Nx],
            ↪  x[nw, nx, 1, 1]       == x0[nw, nx]
188         Cont_Eq_rest[nw in 1:Nw,     nx in 1:Nx,  nfe in 2:NFE],
            ↪  x[nw, nx, nfe, 1]     == x[nw, nx, nfe-1, end]
189     end)
190     ##constraint on input=================================================================
191     @NLconstraints(m1, begin
192         #Defining Inequality Constraints in each line
193         # Constr_Inp1[nw in 1:Nw], abs((_gl[nw, 1] - u0[nw]) / (_gl[nw, 1]))       <= 0.5
194         # Constr_Inp2[nw in 1:Nw, nfe in 2:NFE], abs((_gl[nw, nfe] - _gl[nw, nfe-1]) /
            ↪  (_gl[nw, nfe]))       <= 1
195         #Constr_Inp1[nfe in 1:NFE-1], abs((_gl[1, nfe+1] - _gl[1, nfe]) / (_gl[1,nfe]))
            ↪  <= 0.5
196         #Constr_Inp2[nfe in 1:NFE-1], abs((_gl[2, nfe+1] - _gl[2, nfe]) / (_gl[2,nfe]))
            ↪  <= 0.5
197         #Constr_Inp3[nfe in 1:NFE-1], abs((_gl[3, nfe+1] - _gl[3, nfe]) / (_gl[3,nfe]))
            ↪  <= 0.5
198                 #Define constraint on maximum handling
199
200         Constr_InpSum[nfe in 1:NFE, ncp=1:NCP], sum(_pg[i, nfe, ncp] for i in 1:Nw)    <=
            ↪  28
201                 #Define constraint on maximum gas available
202
203                 Constr_InpSum2[nfe in 1:NFE], sum((_gl[i, nfe]) for i in 1:Nw)    <=
                    ↪  30
204
205         #In case of more states - pattern
206         #Constr_Ineq999[nfe=1:NFE, ncp=1:NCP], alg[999,nfe,ncp] ==
207     end)
208     ##bjective Function===================================================================
209     # @NLobjective(m1, Min,  -(sum(ρ * _po[nw, nfe, end] - _gl * _gl[nw, nfe] for nw in
        ↪  1:Nw, nfe in 1:NFE) -  * sum((_gl[nfe+1] - _gl[nfe])^2 for nw in 1:Nw, nfe in
        ↪  1:NFE-1)))
210     @NLobjective(m1, Min ,  -(ρ * sum(_po[nw, nfe, ncp] for nw in 1:Nw, nfe in 1:NFE, ncp
        ↪  in 1:NCP) - _gl * sum(_gl[nw, nfe] for nw in 1:Nw, nfe in 1:NFE) -   *
        ↪  sum(((_gl[nw, nfe+1] - _gl[nw, nfe])^2) for nw in 1:Nw, nfe in 1:NFE-1) -
        ↪  *sum((_gl[nw, 1]-u0[nw,end])^2 for nw in 1:Nw)))
211     #@NLobjective(m1, Min,  -sum(ρ * _po[nw, nfe, end] - _gl * _gl[nw, nfe] for nw in
        ↪  1:Nw, nfe in 1:NFE) )
212
213     ##=======================================================
214         optimize!(m1)
215     JuMP.termination_status(m1)
216     JuMP.solve_time(m1)
217     ##-------------------------------------------------------
218     return JuMP.value.(u)[:,1,1:t_min_inp * 15 + 1], JuMP.value.(_po)[:, 1:t_min_inp * 15
        ↪  + 1, end], JuMP.value.(_pg)[:, 1:t_min_inp * 15 + 1, end], objective_value(m1)
219     # return JuMP.value.(u)[:,1,:],
220     #        JuMP.value.(z)[:, 10, :, end],
221     #        JuMP.value.(z)[:, 9, :, end]
222 end
```

D Collocation.jl

```
function Collocation_Matrix(N)

      #Radau
      if  N == 2

      t1 = 0.3333333
      t2 = 1.0

      M1 = [
            t1 1 / 2 * t1^2
            t2 1 / 2 * t2^2
            ]
      M2 = [
            1 t1
            1 t2
            ]

      M = M1 * inv(M2)

      elseif  N == 3

            t1 = 0.155051
            t2 = 0.644949
            t3 = 1.0

            M1 = [
                  t1 1 / 2 * t1^2 1 / 3 * t1^3
                  t2 1 / 2 * t2^2 1 / 3 * t2^3
                  t3 1 / 2 * t3^2 1 / 3 * t3^3
                  ]
            M2 = [
                  1 t1 t1^2
                  1 t2 t2^2
                  1 t3 t3^2
                  ]

            M = M1 * inv(M2)

      elseif N == 4

            t1 = 0.088588;
            t2 = 0.409467;
            t3 = 0.787659;
            t4 = 1;


            M1 = [
                  t1 1 / 2 * t1^2 1 / 3 * t1^3 1 / 4 * t1^4
                  t2 1 / 2 * t2^2 1 / 3 * t2^3 1 / 4 * t2^4
                  t3 1 / 2 * t3^2 1 / 3 * t3^3 1 / 4 * t3^4
                  t4 1 / 2 * t4^2 1 / 3 * t4^3 1 / 4 * t4^4
                  ]
            M2 = [
                  1 t1 t1^2 t1^3
                  1 t2 t2^2 t2^3
                  1 t3 t3^2 t3^3
                  1 t4 t4^2 t4^3
                  ]

            M = M1 * inv(M2)
```

```
61
62      elseif N == 5
63
64            t1 = 0.057104;
65            t2 = 0.276843;
66            t3 = 0.583590;
67            t4 = 0.860240;
68            t5 = 1;
69
70            M1 = [
71                  t1 1 / 2 * t1^2 1 / 3 * t1^3 1 / 4 * t1^4  1 / 5 * t1^5
72                  t2 1 / 2 * t2^2 1 / 3 * t2^3 1 / 4 * t2^4  1 / 5 * t2^5
73                  t3 1 / 2 * t3^2 1 / 3 * t3^3 1 / 4 * t3^4  1 / 5 * t3^5
74                  t4 1 / 2 * t4^2 1 / 3 * t4^3 1 / 4 * t4^4  1 / 5 * t4^5
75                  t4 1 / 2 * t4^2 1 / 3 * t4^3 1 / 4 * t4^4  1 / 5 * t5^5
76                  ]
77            M2 = [
78                  1 t1 t1^2 t1^3 t1^4
79                  1 t2 t2^2 t2^3 t2^4
80                  1 t3 t3^2 t3^3 t3^4
81                  1 t4 t4^2 t4^3 t4^4
82                  1 t4 t4^2 t4^3 t5^4
83                  ]
84
85            M = M1 * inv(M2)
86
87      end
88
89
90      return M
91 end
```

E InitiStates.jl

```
1  function Initial_States(U_opt, Nw, t_min, t_min_inp)
2      # plotlyjs()
3      include("Plant.jl")
4      ##
5
6      # t1 = 2
7      # t2 = 2
8      # t_min = t1+t2
9
10     # U_opt=hcat(4*ones(Nw, 15*t1 + 1), 4*ones(Nw, 15 * t2 + 1 ))
11
12     # dicc = Dict(
13     #     :x0       => [ 0.402602  0.592004  1.3199
14     #                    0.5657    0.7424    1.7786
15     #                    0.6603    0.7065    3.3344]
16     # )
17
18     dicc = Dict(
19         :x0       => [ 1.32 0.8 6.0
20                        1.32 0.8 6.0
21                        1.32 0.8 6.0]
22     )
23     ##
24     Full_states = Plant(Nw, U_opt, dicc, t_min)
25
```

```
    myDict = Dict(
        :m_ga_pro   => Full_states[:m_ga_pro][:,end - (15*t_min_inp):end],
        :m_gt_pro   => Full_states[:m_gt_pro][:,end - (15*t_min_inp):end],
        :m_ot_pro   => Full_states[:m_ot_pro][:,end - (15*t_min_inp):end],
        :U_pro      => Full_states[:U_pro   ][:,end - (15*t_min_inp):end],
        :P_a0       => Full_states[:P_a0    ][:,end - (15*t_min_inp):end],
        :_a0       => Full_states[:_a0    ][:,end - (15*t_min_inp):end],
        :P_wh0      => Full_states[:P_wh0   ][:,end - (15*t_min_inp):end],
        :_m0       => Full_states[:_m0    ][:,end - (15*t_min_inp):end],
        :_pc0      => Full_states[:_pc0   ][:,end - (15*t_min_inp):end],
        :P_wi0      => Full_states[:P_wi0   ][:,end - (15*t_min_inp):end],
        :_iv0      => Full_states[:_iv0   ][:,end - (15*t_min_inp):end],
        :_pg0      => Full_states[:_pg0   ][:,end - (15*t_min_inp):end],
        :_po0      => Full_states[:_po0   ][:,end - (15*t_min_inp):end],
        :P_bh0      => Full_states[:P_bh0   ][:,end - (15*t_min_inp):end],
        :_ro0      => Full_states[:_ro0   ][:,end - (15*t_min_inp):end],
        :_rg0      => Full_states[:_rg0   ][:,end - (15*t_min_inp):end],
        :dx0        => Full_states[:dx0     ],
        :x0         => Full_states[:x0      ],
        :dx1        => Full_states[:dx1     ][:,end - (15*t_min_inp):end],
        :dx2        => Full_states[:dx2     ][:,end - (15*t_min_inp):end],
        :dx3        => Full_states[:dx3     ][:,end - (15*t_min_inp):end]
    )
    return myDict
end
```

F Parameters.jl

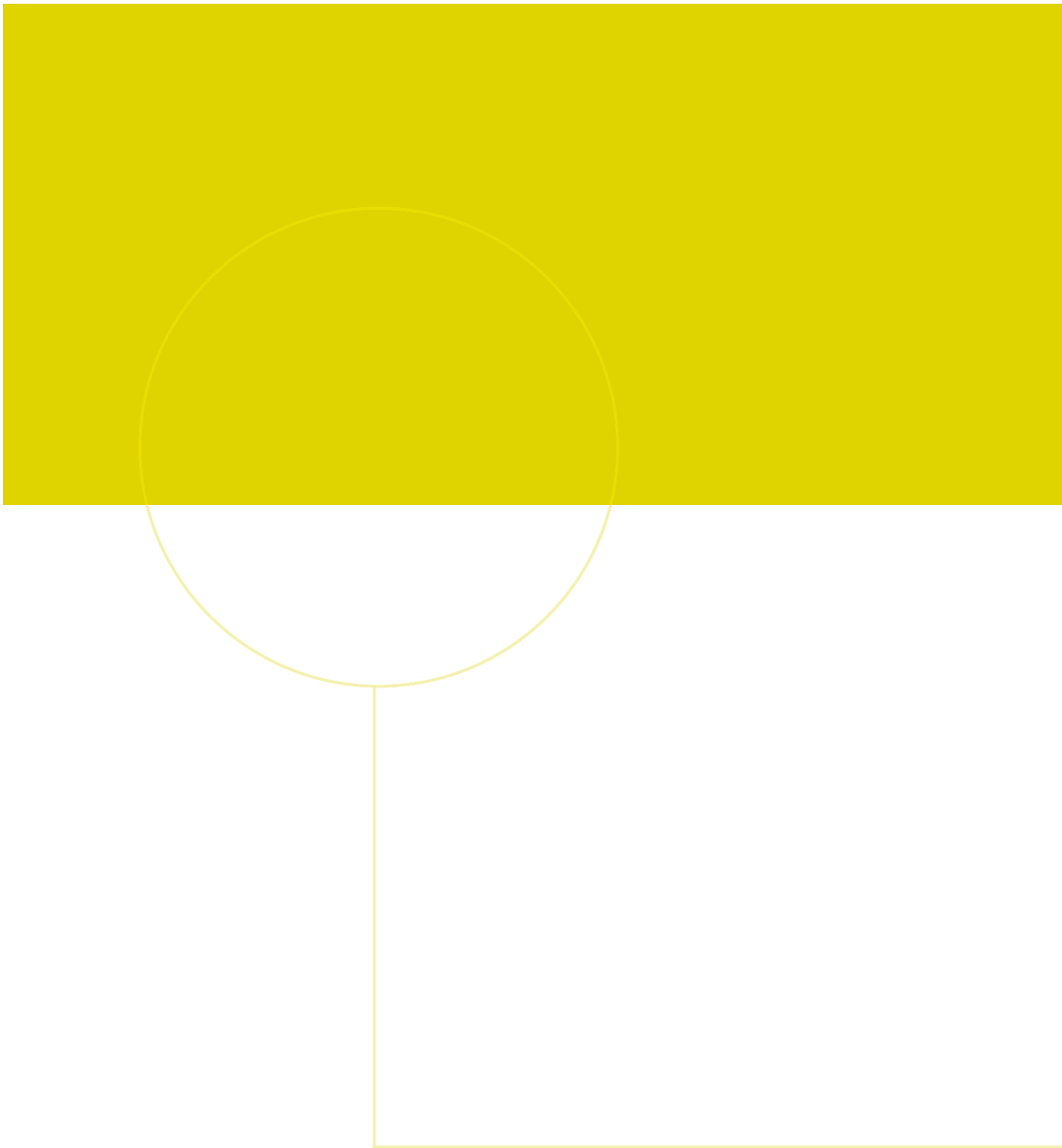
```
@with_kw struct WellPar1
    # L, H, D = Length, Height, Diameter [m]
    # well
    L_w = 1500 # length well
    H_w = 1000 # height well
    D_w = 0.121 # diameter well
    # bottom hole
    L_bh = 500
    H_bh = 500
    D_bh = 0.121
    # annulus
    L_a =  L_w
    H_a = H_w
    D_a = 0.189

     = 800 # density of oil, kg/m3
    mu_oil = 1*0.001

    C_iv = 1e-3 # injection valve characteristic, m2
    C_pc = 2e-3 # choke valve characteristic, m2

    GOR = 0.1 # Gas Oil Ratio
    Press_r =  150 # reservoir pressure
    PI = 2.2 # Productivity index, kg/s/bar
    P_m = 20  # manifold pressure
    T_a = 28 + 273 # K
    T_w = 32  + 273 # K

    Mw = 20e-3 # ? units kg/mol
    #Other Parameter
    A_bh = *D_bh^2/4
```

```
    LA_bh  = L_bh*A_bh
    A_w = *D_w^2/4
    LwAw  = L_w*A_w
    g = 9.81 # m/s2
    R = 8.314 # J/mol K
    V_a = L_a*(*(D_a/2)^2 - *(D_w/2)^2)
end

@with_kw struct WellPar2
    # L, H, D = Length, Height, Diameter [m]
    # well
    L_w = 1500 .*ones(2) # length well
    H_w = 1000 .*ones(2) # height well
    D_w = 0.121 .*ones(2) # diameter well
    # bottom hole
    L_bh = 500 .*ones(2)
    H_bh = 500 .*ones(2)
    D_bh = 0.121 .*ones(2)
    # annulus
    L_a =  L_w .*ones(2)
    H_a = H_w .*ones(2)
    D_a = 0.189 .*ones(2)
     = 800 .*ones(2) # density of oil, kg/m3
    mu_oil = 1*0.001 .*ones(2)

    C_iv = 1e-3 .*ones(2) # injection valve characteristic, m2
    C_pc = 2e-3 .*ones(2) # choke valve characteristic, m2

    GOR = 0.1 .*ones(2) # Gas Oil Ratio
    Press_r = 150 .*ones(2) # reservoir pressure
    PI = 2.2 .*ones(2) # Productivity index, kg/s/bar
    P_m = 20 .*ones(2) # manifold pressure
    T_a = (28 + 273) .*ones(2) # K
    T_w = (32  + 273) .*ones(2) # K

    Mw = 20e-3 .*ones(2) # ? units kg/mol
    #Other Parameter
    A_bh = (.*D_bh.^2/4) .*ones(2)
    LA_bh  = (L_bh.*A_bh) .*ones(2)
    A_w = (.*D_w.^2/4) .*ones(2)
    LwAw  = (L_w.*A_w) .*ones(2)
    g = 9.81 .*ones(2) # m/s2
    R = 8.314 .*ones(2) # J/mol K
    V_a = (L_a.*(.*(D_a./2).^2 - .*(D_w./2).^2) ) .*ones(2)
end

@with_kw struct WellPar3
    # L, H, D = Length, Height, Diameter [m]
    # well
    L_w = [1500.0, 1500.0, 1500.0] # length well
    H_w = [1000.0, 1000.0, 1000.0] # height well
    D_w = [0.121, 0.121, 0.121]  # diameter well
    # bottom hole
    L_bh = [500.0, 500.0, 500.0]
    H_bh = [500.0, 500.0, 500.0]
    D_bh = [0.121, 0.121, 0.121]
    # annulus
    L_a =  [1500.0, 1500.0, 1500.0]
    H_a = [1000.0, 1000.0, 1000.0]
    D_a = [0.189, 0.189, 0.189]

     = [800.0, 800.0, 800.0]   # density of oil, kg/m3
    mu_oil = [0.001, 0.001, 0.001]
```

```
    C_iv = [1e-3, 1e-3, 1e-3]  # injection valve characteristic, m2
    C_pc = [2e-3, 2e-3, 2e-3] # choke valve characteristic, m2

    GOR = [0.1, 0.1, 0.1] # Gas Oil Ratio
    Press_r = [150.0, 150.0, 150.0] # reservoir pressure
    PI = [2.2, 2.2, 2.2] # Productivity index, kg/s/bar
    P_m = [20.0, 20.0, 20.0]  # manifold pressure
    T_a = [301.0, 301.0, 301.0]   # K
    T_w = [305.0, 305.0, 305.0]  # K

    Mw = [0.02, 0.02, 0.02]  # Mulecular weight units kg/mol
    #Other Parameter
    A_bh = .*(D_bh./2).^2
    LA_bh  = L_bh.*A_bh
    A_w = (.*D_w.^2/4)
    LwAw  = L_w.*A_w
    g = [9.81, 9.81, 9.81] # m/s2
    R = [8.314, 8.314, 8.314] # J/mol K
    V_a = L_a.*(.*(D_a./2).^2 - .*(D_w./2).^2)
end

@with_kw struct WellPar3diff
    # L, H, D = Length, Height, Diameter [m]
    # well
    L_w = [1500.0, 1500.0, 1500.0] # length well
    H_w = [1000.0, 1000.0, 1000.0] # height well
    D_w = [0.121, 0.121, 0.121]  # diameter well
    # bottom hole
    L_bh = [500.0, 500.0, 500.0]
    H_bh = [500.0, 500.0, 500.0]
    D_bh = [0.121, 0.121, 0.121]
    # annulus
    L_a =  [1500.0, 1500.0, 1500.0]
    H_a = [1000.0, 1000.0, 1000.0]
    D_a = [0.189, 0.189, 0.189]

     = [800.0, 800.0, 800.0]   # density of oil, kg/m3
    mu_oil = [0.001, 0.001, 0.001]
    C_iv = [1e-3, 1e-3, 1e-3]  # injection valve characteristic, m2
    C_pc = [2e-3, 2e-3, 2e-3] # choke valve characteristic, m2

    GOR = [0.1, 0.1, 0.1] # Gas Oil Ratio
    Press_r = [150.0, 150.0, 150.0] # reservoir pressure
    PI = [1.2, 2.2, 3.2] # Productivity index, kg/s/bar
    P_m = [20.0, 20.0, 20.0]  # manifold pressure
    T_a = [301.0, 301.0, 301.0]   # K
    T_w = [305.0, 305.0, 305.0]  # K

    Mw = [0.02, 0.02, 0.02]  # Mulecular weight units kg/mol
    #Other Parameter
    A_bh = .*(D_bh./2).^2
    LA_bh  = L_bh.*A_bh
    A_w = (.*D_w.^2/4)
    LwAw  = L_w.*A_w
    g = [9.81, 9.81, 9.81] # m/s2
    R = [8.314, 8.314, 8.314] # J/mol K
    V_a = L_a.*(.*(D_a./2).^2 - .*(D_w./2).^2)
end

@with_kw struct WellPar3diffp5
    # L, H, D = Length, Height, Diameter [m]
    # well
    L_w = [1500.0, 1500.0, 1500.0] # length well
```

```
    H_w = [1000.0, 1000.0, 1000.0] # height well
    D_w = [0.121, 0.121, 0.121]  # diameter well
    # bottom hole
    L_bh = [500.0, 500.0, 500.0]
    H_bh = [500.0, 500.0, 500.0]
    D_bh = [0.121, 0.121, 0.121]
    # annulus
    L_a =  [1500.0, 1500.0, 1500.0]
    H_a = [1000.0, 1000.0, 1000.0]
    D_a = [0.189, 0.189, 0.189]

     = [800.0, 800.0, 800.0]   # density of oil, kg/m3
    mu_oil = [0.001, 0.001, 0.001]
    C_iv = [1e-3, 1e-3, 1e-3]  # injection valve characteristic, m2
    C_pc = [2e-3, 2e-3, 2e-3] # choke valve characteristic, m2

    GOR = [0.1, 0.1, 0.1] # Gas Oil Ratio
    Press_r = [150.0, 150.0, 150.0] # reservoir pressure
    PI = [1.26, 2.31, 3.36] # Productivity index, kg/s/bar
    P_m = [20.0, 20.0, 20.0]  # manifold pressure
    T_a = [301.0, 301.0, 301.0]   # K
    T_w = [305.0, 305.0, 305.0]  # K

    Mw = [0.02, 0.02, 0.02]  # Mulecular weight units kg/mol
    #Other Parameter
    A_bh = .*(D_bh./2).^2
    LA_bh  = L_bh.*A_bh
    A_w = (.*D_w.^2/4)
    LwAw  = L_w.*A_w
    g = [9.81, 9.81, 9.81] # m/s2
    R = [8.314, 8.314, 8.314] # J/mol K
    V_a = L_a.*(.*(D_a./2).^2 - .*(D_w./2).^2)
end

@with_kw struct WellPar3diffm5
    # L, H, D = Length, Height, Diameter [m]
    # well
    L_w = [1500.0, 1500.0, 1500.0] # length well
    H_w = [1000.0, 1000.0, 1000.0] # height well
    D_w = [0.121, 0.121, 0.121]  # diameter well
    # bottom hole
    L_bh = [500.0, 500.0, 500.0]
    H_bh = [500.0, 500.0, 500.0]
    D_bh = [0.121, 0.121, 0.121]
    # annulus
    L_a =  [1500.0, 1500.0, 1500.0]
    H_a = [1000.0, 1000.0, 1000.0]
    D_a = [0.189, 0.189, 0.189]

     = [800.0, 800.0, 800.0]   # density of oil, kg/m3
    mu_oil = [0.001, 0.001, 0.001]
    C_iv = [1e-3, 1e-3, 1e-3]  # injection valve characteristic, m2
    C_pc = [2e-3, 2e-3, 2e-3] # choke valve characteristic, m2

    GOR = [0.1, 0.1, 0.1] # Gas Oil Ratio
    Press_r = [150.0, 150.0, 150.0] # reservoir pressure
    PI = [1.14, 2.09, 3.04] # Productivity index, kg/s/bar
    P_m = [20.0, 20.0, 20.0]  # manifold pressure
    T_a = [301.0, 301.0, 301.0]   # K
    T_w = [305.0, 305.0, 305.0]  # K

    Mw = [0.02, 0.02, 0.02]  # Mulecular weight units kg/mol
    #Other Parameter
```

```
    A_bh = .*(D_bh./2).^2
    LA_bh  = L_bh.*A_bh
    A_w = (.*D_w.^2/4)
    LwAw  = L_w.*A_w
    g = [9.81, 9.81, 9.81] # m/s2
    R = [8.314, 8.314, 8.314] # J/mol K
    V_a = L_a.*(.*(D_a./2).^2 - .*(D_w./2).^2)
end
```



NTNU

Norwegian University of
Science and Technology