NTNU
Department of Chemical Engineering

TKP4580 - Specialization project

Implementation of health-aware MPC-controller in a gas-lifted well network

Author:
Salmon Yemane Ghebredngl

Supervisor:
Johannes jäschke

Co-supervisor:
Jose Otavio Assumpcao Matias

December 29, 2020

Contents

List of Figures

List of Tables

Abstract

This project presents the development of a model predictive controller (MPC), that incorporates health monitoring and diagnostics of choke valves to optimize the production of oil in a gas-lifted network. Data-driven approach was used to implement diagnostics of the choke valves. An existing model of the gas-lifted well network with three wells and a phenomenological erosion model of the choke valves was used in an MPC control framework, to maximize the total oil production while keeping the erosion of the choke valves on the three wells below 2mm. The erosion was estimated using a simple regression. The results obtained from the MPC are promising and were able to successfully optimize the production while keeping the erosion of the choke valves below the set threshold value. Further research should look in to applying and testing this control structure in an experiment to further validate the possibility of using data-driven models as a diagnostic tool in the control structure.

Preface

This project was done as part of the completion of the master of science in Chemical Engineering at the Norwegian University of Science and Technology (NTNU). It presents the development of a model predictive controller (MPC) to maximize production while preventing severe degradation of critical components in a gas-lifted network.

It has been a very interesting experience to work with this nascent technology and follow the current researches and developments of control frameworks. Furthermore, I would like to express my greatest gratitude to my supervisor Associate Professor Johannes Jäschke and co-supervisor Jose Otavio Assumpcao Matias for their valuable guidance, assistance and support during the whole process of this project. Special thanks to Jose Otavio Assumpcao Matias for always being available for questions and taking the time for me in his hectic schedule.

1 Introduction

1.1 Background

Companies in the oil and gas industry often have to deal with the intuitive trade-off between optimizing production and minimizing equipment degradation. In oil wells, for example, increasing the oil production is often accompanied by the increase of equipment wear. Process engineers are therefore often forced to adopt conservative production strategies, leading to sub-optimal operation and potential profit loss. In addition, the oil and gas industry transports oil and gas across continents using high-pressure steel pipelines that must operate for decades without failure, so that neither the sea waters nor the air is unnecessarily contaminated[5]. Process control and optimization is at the core of operating those processes successfully as it enables not only safe operation, but also maximum performance utilization.

Equipment reliability is even becoming ever more important to the oil and gas industry, due to the liability issues that occurs when reliability is not assured. Unplanned maintenance interventions are often very expensive and must be minimized at all cost. Understanding and forecasting the degradation process of an equipment is fundamental in mitigating these costs. However, the production of oil and gas is a hugely complicated process with various technical challenges, among them being optimizing plant performance to achieve maximum profit. Failure and degradation of equipment's have been one of the drawbacks of increasing performance. Specially, erosion due to sand production have plagued the oil and gas industry for decades. Modern practices of oil extraction have made sand production even more prevalent. Although, engineers over the years have been trying to mitigate equipment erosion using many strategies and innovative approaches, it has certainly not been good enough to ensure reliability and prevent failures.

1.2 Literature review

Traditionally, most of the modelling in chemical processes is done using first principle models that are prone to inaccuracy due to the complexity of the problem[6]. The erosion rate of a subsea choke valves is dependent on uncertain parameters, for example the stream composition, the impact velocity and the angle of fluid particles in contact with the equipment's surface. Empirical data-driven models in contrast do bot depend on uncertain parameters as they are derived from data, which can be either extracted from simulation of models or from field measurements.

Currently, control strategies that consider diagnostics and prognostics into the control structure remain little explored. The first known usage of prognostics in a control structure is found in paper by T.Escobet, V.Puig and F.Nejjari[7]. This paper tries to integrate control and prognosis where, a conveyor belt that uses an AC electrical motor to move a cart from one end to the other end is used as the system. This new method based on both current and future health state estimates, provided by a prognosis module, takes into account the systems health information in the control objectives. The objective was to extend the useful lifetime of the conveyor belt by adjusting set-points to a simple PID controller.

More case studies that rely on advanced control techniques have also been explored. A research paper by H. Sanchez, T.Escobet, V.Puig and peter Fogh Odegaard pre-

sented the use of an advanced control technique, MPC, integrated with fatigue-based prognosis approach to minimize the damage of wind. turbines [8]. Another paper by A.VerheyleweghenJulie, M. Gjøby, J.Jäschke studied the use of a health-aware Robost MPC for a sub sea compression system subject to degradation. In this paper a hierarchical approach was used for operating compressors subject to degradation. The degradation was estimated using Paris Law, one of the most used models to describe crack propagation in systems subject to stress, with a corrective online parameter estimation [9].

1.3 Objectives

The main scope of this project thesis is to investigate the development of a Health-Aware Model predictive controller, which incorporates diagnostics in to the control structure, applied to a real sub-sea oil and gas production network. More precisely, this projects aims to combine both process control and diagnostics.

For this purpose the following specific objectives were specified:

- Analyze the model of the the sub-sea oil and gas production network
- Identify linear models based on simulated data from the model
- Apply model predictive controller to this process, using the linear model identified
- Study two case studies, one with constant sand production rate and one with exponentially varying sand production rate

1.4 Simulation setting

The proposed control structure is presented in figure 1.1, illustrating the integration of health monitoring, diagnostics and control. Instead of the true erosion being fed to the controller, the estimated erosion through data-driven methods will be fed to the controller.

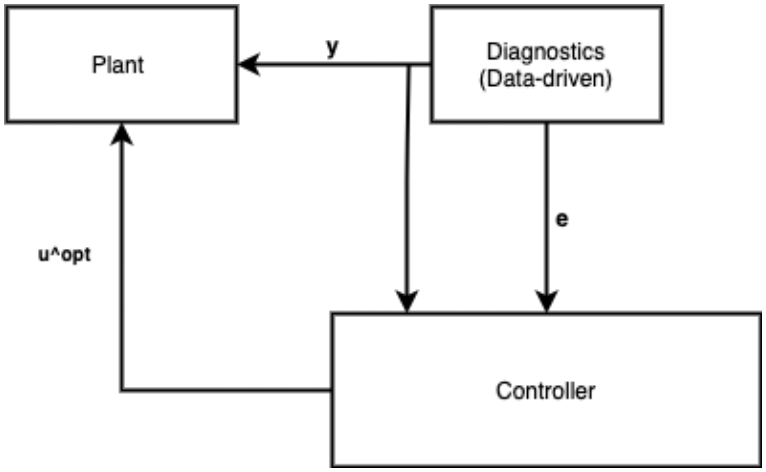

Figure 1.1: Simple Illustration of control structure that incorporates diagnostics (Drawn in drawio)

2 Theory

This chapter introduces the statistical methods that will be used in the modelling and analysis of simulated data. The first step of this process is pre-processing where normalization will be used. Then multiple linear regression will be applied to the data to estimate the erosion rate. Thereafter, an advanced control technique MPC, will be covered along with methods to solve algebraic differential equations.

2.1 Empirical Modeling

Empirical modeling has been a useful tool for the analysis of various problems across a number of field of study [10].This type of modeling is particularly helpful when parametric models can not be constructed due to a number of reasons. Based on different methods and approaches, empirical modeling enables the user to obtain an initial understanding of the relationships that exists among the different variables belonging to a particular process. During the process of empirical modeling, it is always desirable to perform both initial(data cleaning and data screening) and confirmatory analyses (variance and regression analysis) of the available data at hand. This ensures that the model being developed is close to the real model. However, it is not always possible to do confirmatory analyses. This means that oftentimes than not, the user have to make decisions about which variable to include in the modeling based only on the results from the initial models. This part of the chapter describes, the mathematical background of developing such empirical models which is to be used in the estimation of the erosion.

2.2 Data Pre-Processing

Data pre-processing is a widely used technique that involves transforming raw data into another format before analysis. There exists several methods such as centering and scaling so that no single variable dominates the system due to its large scale and variance. Furthermore, normalization can also be used when dealing with different scales. Another method which can be applied is principal component analysis, which is used when the dimension of the data available is large and reduction of the data without losing information is needed to better manage the analysis. In this project, normalization is chosen due to the varying orders of magnitude and units of measurements in the data.

2.2.1 Normalization

Normalization is used when the data consists of variables with different scales. This process compensates the variability in the orders of magnitude and units of measurements in the data by scaling all the data to be centered with unit variance and mean of zero. This is implemented using the standard score formula:

$$Z = \frac{X - \mu}{\sigma} \tag{2.1}$$

Where Z is the standard score, X is the original data value, μ is the mean.

2.3 Regression Model

The basic idea of regression analysis is to obtain a model for the functional relationship between a response variable (often referred to as the dependent variable) and one or more predictor variables (often referred to as the independent variables) [11]. This model provides both the ability to determine the predictor variables that affects the response variable and the ability to find out what happens to the response variable for specific changes in the predictor variables. For example, financial officers must predict future cash flows based on some values of interest rates, raw material costs, salary increases, and so on. In designing new training programs for employees, a company may want to study the relationship between employee efficiency and predictor variables such as the results from employment tests, educational background, and previous training. [11]

2.3.1 Linear - Regression

In this project thesis, we consider simple linear regression analysis with multiple predictor variables. Linear regression assumes that there is a linear relationship between a response variable, $\mathbf{Y}$, and a set of predictor variables $\mathbf{X}$ along with some noise ϵ. In this model, the error term ϵ is assumed to be normally distributed, homoscedastic, meaning with the same variance at every X and has mean of zero.

$$\boldsymbol{Y} = \boldsymbol{B}\boldsymbol{X} + \boldsymbol{\epsilon} \tag{2.2}$$

Given a training data we can generate the estimate for B, that is $\hat{B}$. From these estimated parameters, the functional relationship between $\mathbf{Y}$ and $\mathbf{X}$ can be found as follows:

$$\hat{\boldsymbol{Y}} = \hat{\boldsymbol{B}}\boldsymbol{X} \tag{2.3}$$

where $\hat{Y}$ and $\hat{B}$ are the estimates of the true values. The least squares estimate for the regression coefficients in multiple linear regression in matrix form is given by:

$$\hat{\boldsymbol{B}} = (\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X})^{-1}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{Y} \tag{2.4}$$

2.4 Stepwise regression

Stepwise regression is a statistical method used to build a model by adding or removing predictor variables, through a series of either F-tests or T-tests. The variables that are added or removed are selected based on the test statistics of the estimated coefficients. There are different approaches to this process such as forward selection and backward elimination. In forward selection method, the model building starts with no variables and variables whose inclusion gives significant improvement of the model fit are added accordingly. While, in backward elimination, the model building is initialized with candidate variables and variables whose loss gives the most insignificant deterioration of the model fit are removed. The advantages of using stepwise regression over other automatic model selection procedures is the ability to manage large amounts of predictor variables and tune the model by selecting the best predictor variables. Furthermore, its faster than other automatic model-selection methods [12], and for those reasons this method is chosen in this project.

2.5 Model Predictive Control

Model predictive control (MPC), is an advanced control technique, used widely in many industrial applications as it offers several advantages. Some of the advantages are its ability to handle non-linear constraints and disturbances. However, this technique is highly dependent on the process model. The availability of a very precise system model is therefore fundamental in using this control technique. [1] The principal idea behind MPC is to solve an optimization problem with a given constraints at each time interval. The aim is to determine a sequence of input moves such that the predicted response tracks a given setpoint. In this method, M control actions are calculated at each time step, and only the first control action is implemented. Once a new measurement is available, the initial condition of the model is updated and a new sequence of control action is again calculated. In this manner, the horizon is displaced towards the future at each instant. This strategy is known as receding strategy and it enables online trucking of unmeasured disturbances. [2]

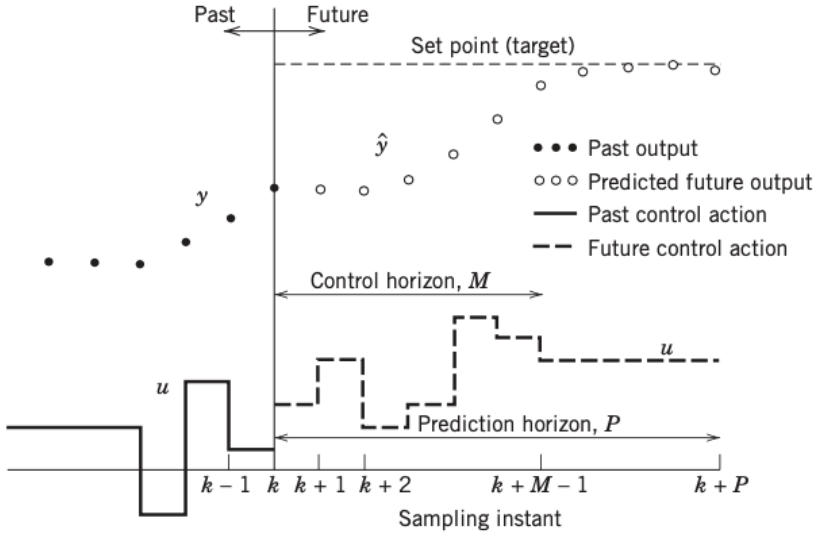

Figure 2.1: Basic concept for model predictive control. [1]

A schematic representation of an MPC controller for SISO system is shown in figure 2.1 with y being the actual output, $\hat{y}$ being the predicted output and u, the manipulated input. At the current sampling instant, denoted by k, the MPC calculates a set of M values of the input $\{u(k+i1), i = 1, 2, ..., M\}$. This set consists of the current input u(k) and $M-1$ future inputs. The input is held constant after the M control moves. The inputs are then calculated so that a set of P predicted outputs $\hat{y}(k+i), i = 1, 2, ..., P\}$ reaches the setpoint. The number of predictions P is referred to as the prediction horizon while the number of calculated control moves M is referred to as the control horizon. [1]

The simple structure of MPC strategy is shown in figure 2.2. As illustrated in the figure, the process model predicts upcoming outputs based on current outputs, previous outputs and the suggested optimal future control actions by the optimizer. [2]

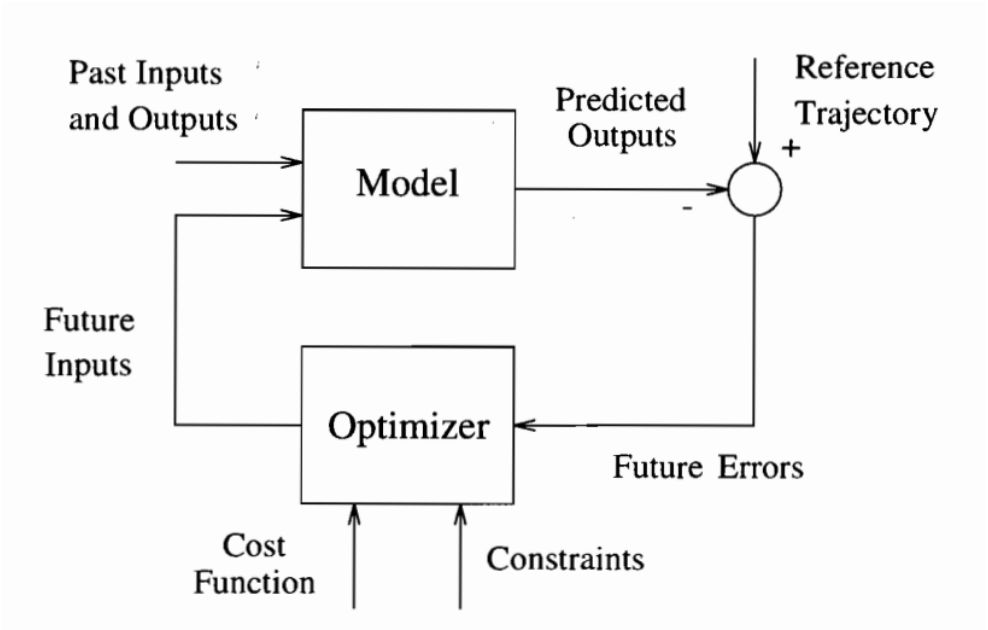

Figure 2.2: Basic structure of MPC [2]

The control calculations are based on optimizing an objective function. There exists different types of objectives for the MPC such as economic control and setpoint control. In this project, an economic objective is considered, where the objective is to minimize an economic cost function. The type of systems we are dealing with consists of both algebraic and differential equations, resulting in a differential algebraic equation system. These equations are used as constraints in the optimization problem. Furthermore, there are also inequality constraints which specify both the allowed values of the inputs and the changes in the inputs. The objective function and the model can therefore be simplified as follows

$$\Psi = \int_0^P (-cost + \frac{1}{2}\Delta u(t)^T R_{\Delta u} \Delta u(t))\, dt \tag{2.5}$$

$$\dot{x} = f(x, z, p, u) \tag{2.6}$$

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

$$0 \leq h(x, z, p, u) \tag{2.8}$$

where P is the prediction horizon, the second term describes a regularization term on the change in inputs,Δu. This forces the controller to minimize the change of inputs. R is a tuning parameter that weights the regularization term in the objective function. $\dot{x}$ describes the set of differential equations, x describes the differential states, z the algebraic states, p are the parameters of the system, u is the input of the system. The model differential equations and the algebraic equations are represented by f and g, respectively.

2.6 Solving Methods for Algebraic Differential Equation

Efficient calculation of large problems requires reliable algorithms. There are two popular options, a sequential approach and a simultaneous approaches. In the sequential approach such as single shooting, the model equations are repeatedly solved by a numerical integrator, which provides the states trajectory and their gradients. The optimization algorithm, then, computes new decision variables and the simulation process is repeated [3]. While, in the simultaneous approaches such as orthogonal collocation the model equations are solved simultaneously with the optimization problem.

Even though, sequential methods are easier to implement, they may use unreasonable time to converge, especially problems with a large numbers of degrees of freedom. The simultaneous methods have generally computational advantage over sequential methods [3]. Especially for control problems with many decision variables and a moderate number of state variables. In this paper, orthogonal collocation has been chosen as the preferred way of solving the dynamic algebraic equation system for its low computational cost and its accurate results.

2.6.1 Orthogonal Collocation

Orthogonal collocation on finite elements is based on dividing the prediction horizon in to finite elements. Each of these elements are then further divided into a given number of collocation points.

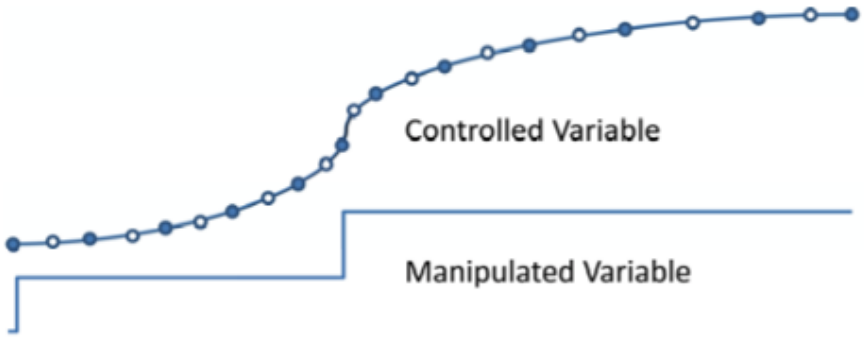

Figure 2.3: Dynamic equations are discretized over a time horizon and solved simultaneously. With one internal node for each segment, this example uses a 2nd order polynomial approximation for each step. [3]

As shown in figure 2.3 the dynamic equations are discretized over a time horizon and solved simultaneously. The solid nodes in the figure represents starting and ending point for local polynomial approximations that are stitched together over the time horizon.[3]. The main idea behind orthogonal collocation is to determine a weighting matrix M that relates the derivatives to non-derivative values over a time horizon at points 1,...,n as exhibited in equation 2.9. The initial value, x_0, is either a fixed initial condition or equal to the final point from the last interval.

$$\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \end{bmatrix} = M \left(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} - \begin{bmatrix} x_0 \\ x_0 \\ x_0 \end{bmatrix} \right) \tag{2.9}$$

The solution of the differential equations at discrete time points is approximated as a polynomial as follows:

$$x(t) = A + Bt + Ct^2 + Dt^3 \tag{2.10}$$

where t is the placement of the collocation points on the finite element. The derivative of x with respect to t is then given by:

$$\dot{x}(t) = Bt + 2Ct + 3Dt^2 \tag{2.11}$$

The collocation points used in this project is the Gauss-Radaue with numbers (0.1151, 0.6449, 1.0000). The time points are shifted to a reference time of zero and final time of 1. This enables the user to calculate the solutions without interpolation. For initial value problems, the coefficients A is equal to x_0, when the initial time is defined as zero. The coefficients B, C, and D are calculated by substituting equation 2.11 into 2.9.

$$M \begin{bmatrix} B + 2Ct_1 + 3Dt_1^2 \\ B + 2Ct_2 + 3Dt_2^2 \\ B + 2Ct_3 + 3Dt_3^2 \end{bmatrix} = \begin{bmatrix} A + Bt + Ct_1^2 + Dt_1^3 \\ A + Bt + Ct_2^2 + Dt_2^3 \\ A + Bt + Ct_3^2 + Dt_3^3 \end{bmatrix} - \begin{bmatrix} x_0 \\ x_0 \\ x_0 \end{bmatrix} \tag{2.12}$$

Rearraging and setting A = x_0 gives:

$$M \begin{bmatrix} 1 + 2t_1 + 3t_1^2 \\ 1 + 2t_2 + 3t_2^2 \\ 1 + 2t_3 + 3t_3^2 \end{bmatrix} \begin{bmatrix} B \\ C \\ D \end{bmatrix} = \begin{bmatrix} t_1 & t_1^2 & t_1^3 \\ t_2 & t_2^2 & t_2^3 \\ t_3 & t_3^2 & t_3^3 \end{bmatrix} \begin{bmatrix} B \\ C \\ D \end{bmatrix} \tag{2.13}$$

Finally, rearranging and solving for M gives the solution:

$$M = \begin{bmatrix} t_1 & t_1^2 & t_1^3 \\ t_2 & t_2^2 & t_2^3 \\ t_3 & t_3^2 & t_3^3 \end{bmatrix} \begin{bmatrix} 1 + 2t_1 + 3t_1^2 \\ 1 + 2t_2 + 3t_2^2 \\ 1 + 2t_3 + 3t_3^2 \end{bmatrix}^{-1} \tag{2.14}$$

For intervals that are not between 0 and 1, a scaling parameter h is introduced.

$$M \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_0 \\ x_0 \\ x_0 \end{bmatrix} + hM \begin{bmatrix} f(x_1, z_1, p_1, u_1) \\ f(x_2, z_2, p_2, u_2) \\ f(x_3, z_3, p_3, u_3) \end{bmatrix} \tag{2.15}$$

The objective function and constraints given in equations 2.5 - 2.8, becomes a non-linear optimization problem which can be solved using orthogonal collocation. A constraint on the differential states are enforced within every collocation point to ensure that the trajectory for the differential states are continuous. The objective function is then evaluated at the end of every collocation point.

3 Process Description

The system studied in this project is a subsea oil and gas production network involving three wells. The wells are connected to a common manifold. The combined flow from the three wells goes through a riser to a topside facility. When the reservoir pressure is no longer high enough to lift the fluids from the reservoir to the top facility, artificial methods are often needed. Among the artificial lifting methods is gas-lift. Artificial gas-lift is added into the fluid mix to reduce mixture density. This decreases the hydrostatic pressure in the bottom, increasing the pressure difference and flow from the reservoir to the top facility. A more detailed description of the system can be found in the paper by A.Verheyleweghen. and supervisor J. Jäschke.[4].

However, increased volume flow leads to an increase in degradation of equipment, like the choke valves in the system. In particular, erosion of choke valves which are used to reduce well pressure and control production is severe. A well stream typically consists of a mix between oil, gas, water, sand and other various particles. When all of these elements hit the internal surface of the choke valves for long periods, it causes erosion which shortens the useful life of the choke valves. Particularly, high production of sand has been known to cause considerable erosion damage in critical parts of the choke valves. Illustration of the gas-lifted subsea oil and gas production system is shown in section 3.1.

3.1 Gas-Lift Model

In this process, the gas is injected at the bottom of the wells through annulus (the void between the piping), decreasing the mixture density of the fluid mix in the tubing. This leads to a lower hydrostatic pressure drop in the well and thereby increasing the pressure difference and flow from the reservoir. The model used to describe this gas-lifted well system is based on the work by Krishnamoorthy used for real-time optimization applied to a gas lifted well system.[13]

The mass balance of the different phases, the density models, the pressure models and the flow models are described as follows.

The mass balance of the wells is given by:

$$\dot{m}_{ga} = w_{gl} - w_{iv} \tag{3.1}$$

$$\dot{m}_{gt} = w_{iv} - w_{pg} + w_{rg} \tag{3.2}$$

$$\dot{m}_{ot} = w_{ro} - w_{po} \tag{3.3}$$

where the m_{ga} is the mass of the gas annulus, m_{gt} is the mass of the gas inside the well tubing, m_{ot} is the mass of oil in the well tubing, w_{gl} is the gas lift injection rate, w_{iv} is the gas flow from the annulus to the tubing, w_{pg} is the flow rate of produced gas, w_{rg} is the flow rate of gas from reservoir and w_{ro} is the flow rate of oil from the reservoir.

These differential equations were set to be algebraic, due to the large time scale difference between the erosion rate and the differential equations, i.e, the equation over can be rewritten as:

$$0 = w_{gl} - w_{iv} \tag{3.4}$$
$$0 = w_{iv} - w_{pg} + w_{rg} \tag{3.5}$$
$$0 = w_{ro} - w_{po} \tag{3.6}$$

The density model is given by:

$$\rho_a = \frac{M_w p_a}{T_a R} \tag{3.7}$$

$$\rho_w = \frac{m_{gt} + m_{ot} - \rho_0 L_r A_r}{L_w A_w} \tag{3.8}$$

where the ρ_a is the density of the gas in the annulus. ρ_w is the density if the fluid mixture in the tubing, M_w is the molecular weight of the gas, R is the gas constant, ρ_0 is the density of the oil in the reservoir, T_a is the annulus temperature, L_r is the length of the well above the injection point and L_w is the length below the injection point, A_r is the cross-sectional area above the injection point and A_w is the cross-sectional area below the injection point.

The pressure model is given by:

$$p_a = (\frac{T_a R}{V_a M_w} + \frac{g L_a}{L_a A_a}) m_{ga} \tag{3.9}$$

$$p_{wh} = \frac{T_w R}{M_w} (\frac{m_{gt}}{L_w A_w + L_r A_r - \frac{m_{ot}}{\rho_0}}) \tag{3.10}$$

$$p_{wi} = p_{wh} + \frac{g}{A_w L_w} (m_{ot} + m_{gt} - \rho_0 L_r A_r) H_W \tag{3.11}$$

$$p_{bh} = p_{wi} + \rho_w g H_r \tag{3.12}$$

In this equations p_a is the annulus pressure, p_{wh} is wellhead pressure. w_{iv} is well injection point pressure, p_{bh} is the bottom hole pressure, L_a is the length of the annulus, A_a is the cross-sectional area of the annulus, T_w is the temperature of the well tubing,H_r is the vertical height of the well tubing below the injection point, H_w is the height of the well tubing above the injection point, g is the gravitational acceleration.

The flow model is given by:

$$w_{iv} = C_{iv} \sqrt{\rho_a max(0, p_{ai} - p_{wi})} \tag{3.13}$$

$$w_{iv} = C_{pc} \sqrt{\rho_w max(0. p_{wh} - p_m)} \tag{3.14}$$

$$w_{pg} = \frac{m_{ot}}{m_{gt} + m_{ot}} w_{pc} \tag{3.15}$$

$$w_{po} = \frac{m_{gt}}{m_{gt} + m_{ot}} w_{pc} \tag{3.16}$$

$$w_{ro} = PI(p_r - p_{bh}) \tag{3.17}$$

$$w_{rg} = GOR * w_{ro} \tag{3.18}$$

where w_{iv} is the gas lift injection valve, w_{pc} is the total flow through the produciton choke, w_{pg} is the produced gas flow rate, w_{po} is the produced oil flow rate, w_{ro} is the reservoir oil flow, w_{rg} is the reservoir gas flow rate, C_{iv} is the injection valve coefficient, C_{pc} is the production choke valve coefficient, PI is the reservoir productivity index, p_r is the reservoir pressure, p_m is the manifold pressure and GOR is the gas-oil ratio. All these models are for a single well which can be easily extended to the three wells.

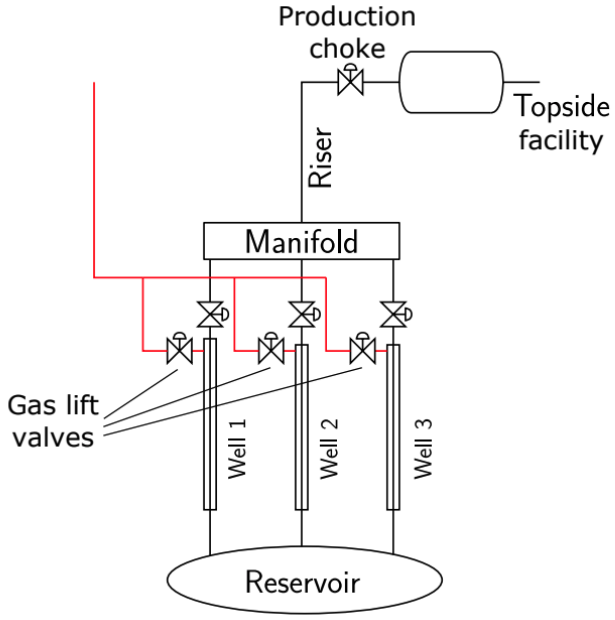

Figure 3.1: Illustration of the gas-lifted network,from the paper by A.Verheyleweghen and jäschke[4]

3.2 Erosion Model

The erosion model used in this project thesis is based on a choke erosion model from DNVGL.[14] The erosion model presented by Verheyleweghen and Jäschke is given by equation 3.19. For the rest of this project thesis, this phenomenological model will be used in the true plant model and the data-driven model will be used to approximate the erosion behavior.

$$\dot{E} = \frac{KF(\alpha)U_p^n}{\rho_t A_t} * G * C_1 * GF * \dot{m}_{sand} * C_{unit} \tag{3.19}$$

Where E is the erosion rate, while K, n, C_1, GF and C_{unit} are constants. $F(\alpha)$ is the ductility and U_p^n is the particle impact velocity. Furthermore, calculation of parameters A and G are shown in the paper by Verheyleweghen and Jäschke.

4 Results and Discussion

The results of this project will be presented in two separate parts. The first part will be focusing on the statistical modelling of the erosion. The goal is to develop an empirical model of erosion to be used for diagnostics in the control structure. While the second part will focus on the performance of the model predictive controller whose main objective is to maximize oil production while keeping the erosion of the three chock valves below 2mm.

All simulations developed in this project were carried out in MATLAB. The script for simulating the process system is based on the equations presented in section 3.1 and the empirical erosion model that is to be generated from measurements. Furthermore, due to the scale of the problem CASADI[15] was used for integrating and optimization the problem. It is an open-source software framework for numerical optimization. It is chosen due to its ability to model solve optimization problems with a large degree of flexibility.

4.1 Data generation for training the data driven models

The data used for estimation of the erosion was generated for both case studies, one with constant sand production rate of 0.01 kg/s and one with exponentially varying sand production rate. This is performed through simulations of the described model in section 1.3. The code that was used for this purpose is shown in appendix 5, listing 2. This simulation of the model creates data for the three wells. The input variable, gas lift rate was used with random variation to provide the different time series of data. In this project, only one time series of data is generated for simplification. The input U, took values between $U_{min} + 1 \cdot (U_{max} - U_{min})$ to $U_{min} + 3 \cdot (U_{max} - U_{min})$. Furthermore, a noise was also added in addition to random variation to incorporate the uncertainty in measurements. The gas lift profile is shown in figure 4.1 and 4.2. As shown in the figures the simulations are run over 500 days iteration, with the gas lift injection changing every 50 days for each time series.

For the second case with varying sand production, an exponential sand production growth was chosen. However, since in the real industry sand production rate is not a continuously measured variable; the models are being fed the true rate sampled every 50 days. As mentioned above in both cases, only one time series is simulated. The measurement data is therefore generated in this manner for each well. This simulation generated data for the control input (gas-lift rate), measurement values of parameters (predictors) and the response (erosion) of each well for 500 days. Since, the models used are trained on the normalized form of this data , the erosion profile was predicted using this training data. The results for the two cases is shown in figure 4.1 and 4.2, respectively.

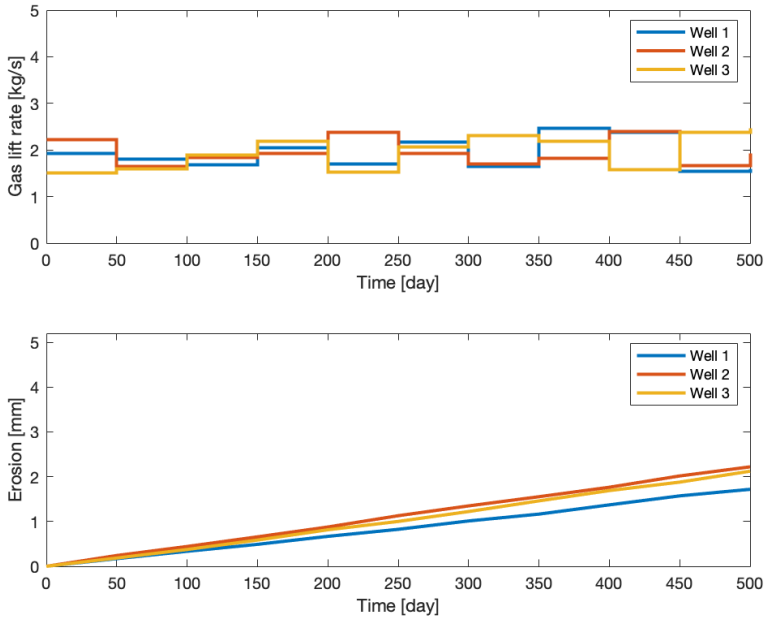

Figure 4.1: Gas-lift rate and erosion in mm plotted against time in days for 3 wells with constant sand production-rate

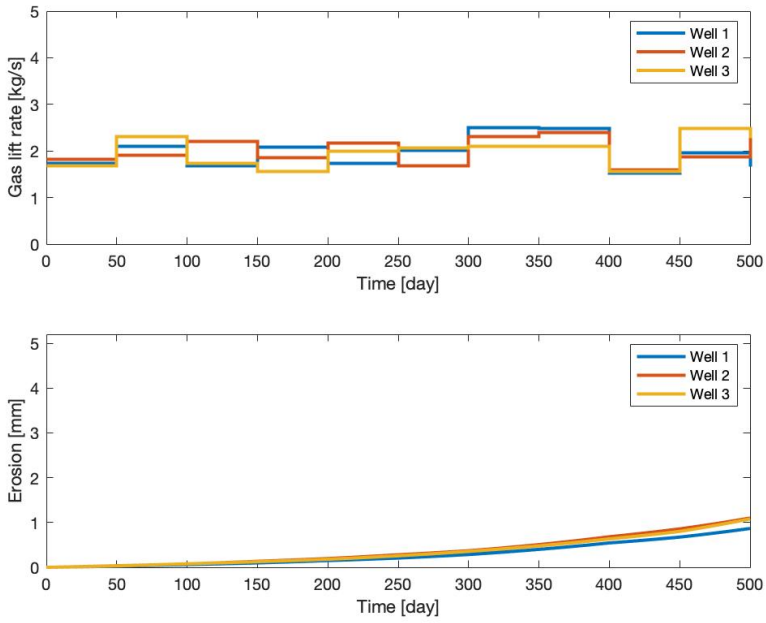

Figure 4.2: Gas-lift rate and erosion in mm plotted against time in days for 3 wells with exponentially varying sand production-rate

4.2 Modelling Erosion using Linear Model

The models used in this project have already been trained and tested by another student. It is therefore important to note that, we are assuming the models are not over-fitting the training data, and they are capable of predicting the erosion reasonably well.

Prior to modelling, the simulated data consisting of 500 measured data points was pre-

processed. The predictor variables shown in table 4.1 were subject to normalization as described in section 2.2.1, such that all the variables have a standard deviation and mean of 1 and 0 respectively. This was done due to the differing units of measurements in the data, as they can give unreasonable results. Furthermore, the models were trained using the gradient of erosion measurements instead of direct erosion measurements, as such, erosion rate was used to predict the erosion, before it was transformed to cumulative erosion.

Table 4.1: The parameters used for the regression, with var 1-9 as predictor variables and var 10 as response variable

predictor variable	Description
var1	Annulus pressure
var2	Well head pressure
var3	Well head oil production rate
var4	Well head gas production rate
var5	Riser head pressure
var6	Manifold pressure
var7	Riser head total oil production rate
var8	Riser head total gas production rate
var9	Gas lift rate
var10	Erosion rate (Response)

A trained linear regression model with interaction terms, which was selected with stepwise regression, was used to predict erosion. The results of the linear regression can be seen in figure 4.3. As can be seen in the figure, the erosion is behaving linearly as expected from the model used to make the simulations with constant sand production rate. During the analysis, the erosion was observed to highly correlate with gas lift rate and flow rate. In addition, it can also be shown that the error percentage of the predicted erosion from the real true erosion was below 1% for the three wells. It can therefore be said that, the model predicts the erosion reasonably well for the case with constant sand production rate.

For the case with varying sand production, the same method was implemented with an additional predictor variable of sand production rate. The difference here is that since the sand production rate is not constant anymore, it has to be fed to the model as the first predictor. This model had therefore, 10 predictor variables instead of 9. However, since in the real world, sand production is not continuously measured variable, the model was fed the true sand production rate sampled every 50 days. The profile as seen in figure 4.4 is an exponential function and the model does worse job here in predicting the erosion, as can be seen from the error percentage.

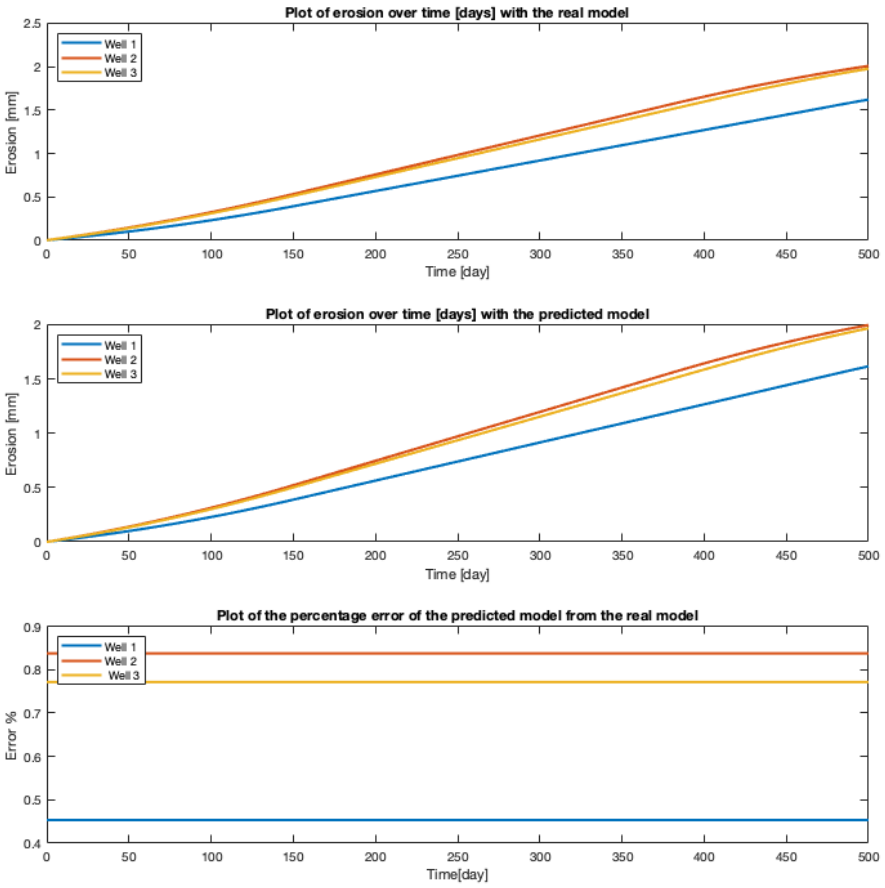

Figure 4.3: Real true erosion vs predicted erosion

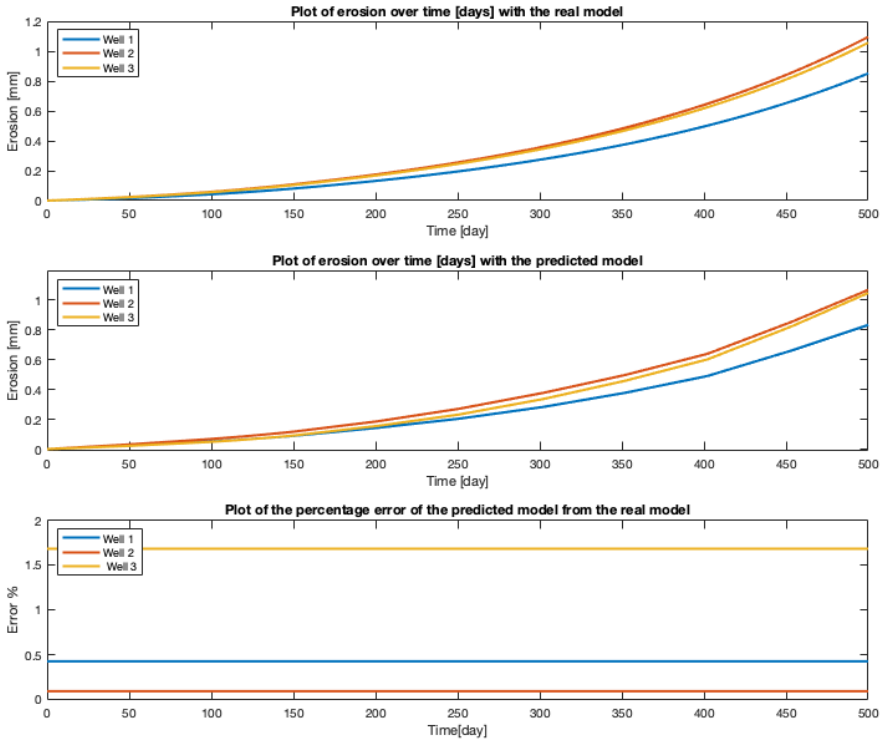

Figure 4.4: Real true erosion vs predicted erosion

4.3 Performance of the MPC controller

The data driven model developed in section 4.2 was used as a model for the MPC. The main objective of the MPC as mentioned previously, is to maximize the total production of oil, while keeping the erosion for each well below a threshold value of 2mm. The differential equations shown in section 3.2 together with the algebraic equations shown in section 3.1 will set the basis for the feasible region and will therefore be the constraints for the optimization problem. The problem for the MPC can then be written as follows:

$$min\Psi = \int_0^P (\sum_{i=1}^{3} -w(t)_{i,po} + \frac{1}{2}\Delta u(t)^T R_{\Delta u}\Delta u(t) + \sum_{i=1}^{3} \rho_{i,s} s_i(t)\, dt \tag{4.1}$$

subject to .

$$\dot{E} = \frac{KF(\alpha)U_p^n}{\rho_t A_t} * G * C_1 * GF * \dot{m}_{sand} * C_{unit} \qquad t \in= [0, P] \tag{4.2}$$

$$g(x) = 0 \qquad t \in= [0, P] \tag{4.3}$$

$$\Delta u(t) = 0 \qquad t \in= [M, P] \tag{4.4}$$

$$-\Delta u_{max} \leq \Delta u(t) \leq \Delta u_{max} \qquad t \in= [0, M] \tag{4.5}$$

$$u_{min} \leq u(t) \leq u_{max} \qquad t \in= [0, P] \tag{4.6}$$

$$0 \leq E(t) + s(t) \leq E_{max} \qquad t \in= [0, P] \tag{4.7}$$

In this optimization problem, $g(x)$ is the algebraic equation mentioned in section 3.1. Furthermore, the term $s_i(t)$ is introduced to the problem as a slack variable to give the

controller flexibility to violate the constraint at high cost for the objective function. The penalty is given by the weighting parameters $\rho_{i,s}$, which was set to be 999999 for the three wells. The main objective of the slack variable is to ensure that the controller does not enter an infeasible region when the controller can no longer satisfy the constraint on the maximum allowed erosion. In addition, M and P represents the control and prediction horizon, respectively. Δu is a regularization term introduced to the problem in case it is ill-conditioned and therefore might make the controller unstable. The weighting of the regularization, $R\Delta u$, is set to be:

$$R\Delta u = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.8}$$

Orthogonal collocation was used to solve the algebraic equations and was executed at every collocation point. An input of 0.4kg/s and an upper limit of $u_{max} = 2kg/s^{-1}$ for the gas lift rate was fed to the system to ensure flow. Furthermore, a conservative value of $\Delta u_{max} = 0.01kg/s^{-1}$ was used in the simulation. The controller was run over a prediction horizon of 100 days and control horizon of 70 day, where each time step is a day. Simulation of the controller for 500 days gave the following results for the two case studies

4.3.1 Case study 1: Constant sand production rate

The simulation results for this case study are shown in figure 4.5. At the beginning of the simulation, the controller tries to maximizes the total production of oil by shooting up the gas lift rate as fast as possible. The controller is however, constrained by the maximum allowed change of input,Δu and therefore reaches a constant region where the gas lift rate can no longer be increased. In this region of the simulation, it is observed that the erosion rate and the total oil production is kept at a constant rate. At around day 380, the controller again adjusts the gas lift rate of well 2 by decreasing, as the systems starts to violate the constraint on the erosion. In addition, the erosion rate of the three wells in the systems was shown to be different, with well 2 being the one exposed to the highest erosion. This can be due to the high ratio of oil to gas in well 2 compared with well 1 and well 3. Furthermore, well 1 was the one with the least erosion due to its low gas to oil ratio and reservoir pressure, and this can be observed in the figure 4.5 as the controller decreases the gas lift of well 1 last. We can conclude that the controller was able to maximize the oil production while keeping the erosion below the threshold which was set to be 2mm.

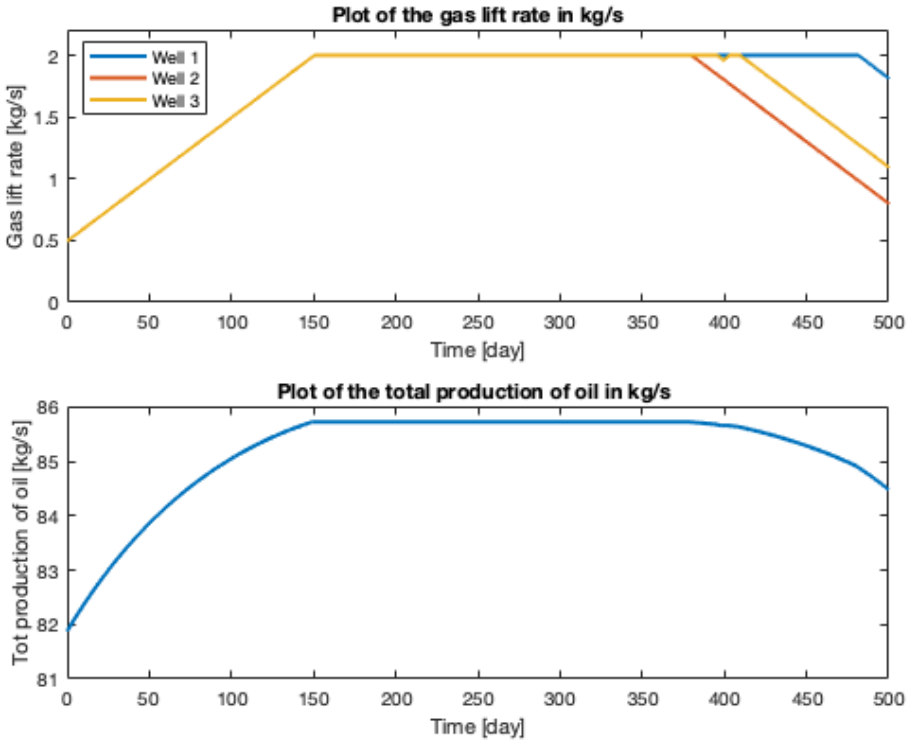

Figure 4.5: Total oil production and gas-lift rate with constant sand production rate

4.3.2 Case study 2: Exponentially varying sand production rate

The goal here is to analyze how the controller behaves when the plant and the controller model are different, i.e there is a plant-model mismatch. The erosion was first estimated using the procedures presented in section 4.2 with a exponentially varying sand production rate. The simulation results of this case study are shown in figures 4.6 and 4.7 with two different initialization for the sand production rate. We observe similar behavior at the beginning of the simulation, where the controller tries to maximize the total production of oil by increasing the gas lift rate as fast as possible. However, the controller shuts of the production earlier in the simulation as the controller starts to violate the maximum allowed erosion, which was set to be 2mm, see 4.6. The sand production gets too high at the end and forces the controller to shut down production and keep the total production of oil at a near zero level. The erosion rate of the three wells as expected was also different here subject to the same arguments in case 1. Well 2 and Well 1 , had the highest and lowest erosion rate respectively. This makes sense as the gas to oil ration of well 2 was the highest and the gas to oil ratio of well 1 was the lowest.

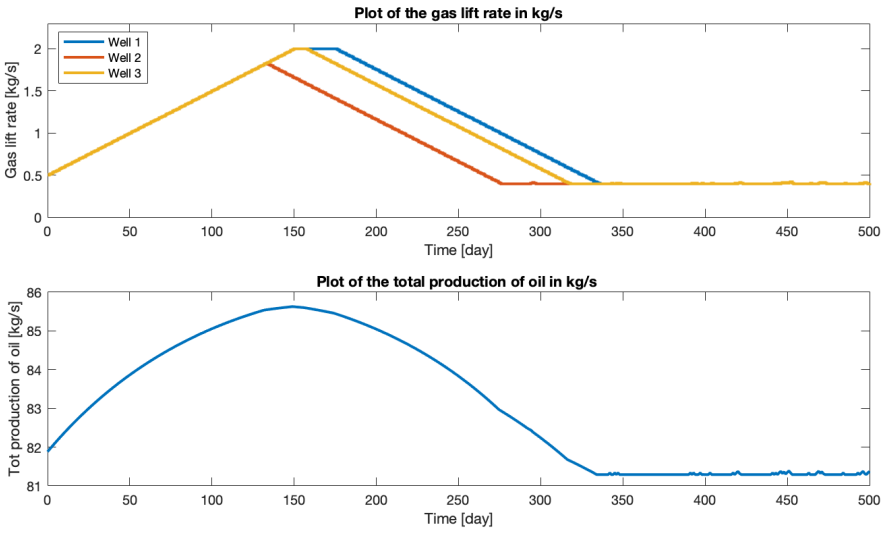

Figure 4.6: Total oil production and gas-lift rate with varying sand production rate.(initialized with a sand production rate of 0.1 at the start)

In the hope of getting a better performance, the sand production rate was initialized with a lower value of 0.01kg/s. The simulation results are shown in 4.7. The controller behaves conservatively and tries to keep a constant erosion rate. Furthermore, the total oil production is also kept at constant in this region. Surprisingly, the erosion rate of the three wells was shown to be the same. It seems that when the sand production rate is no longer constant with a low value, the erosion of the three wells is solely dependent on the sand production rate.

Furthermore, all the controllers were also tested with an increased level of measurement noise, with a random number that is added to the measurements. This showed a degradation on the performance of the controller as the noise level increases. This is expected, as the noise increases, the deviation of the predicted erosion from the true erosion increases and they will worsen the performance of the controller. In this case study, overshoot of sand production rate caused the controller to shut down production. Some kind of controller reconfiguration is therefore needed in order to account for the model mismatch.

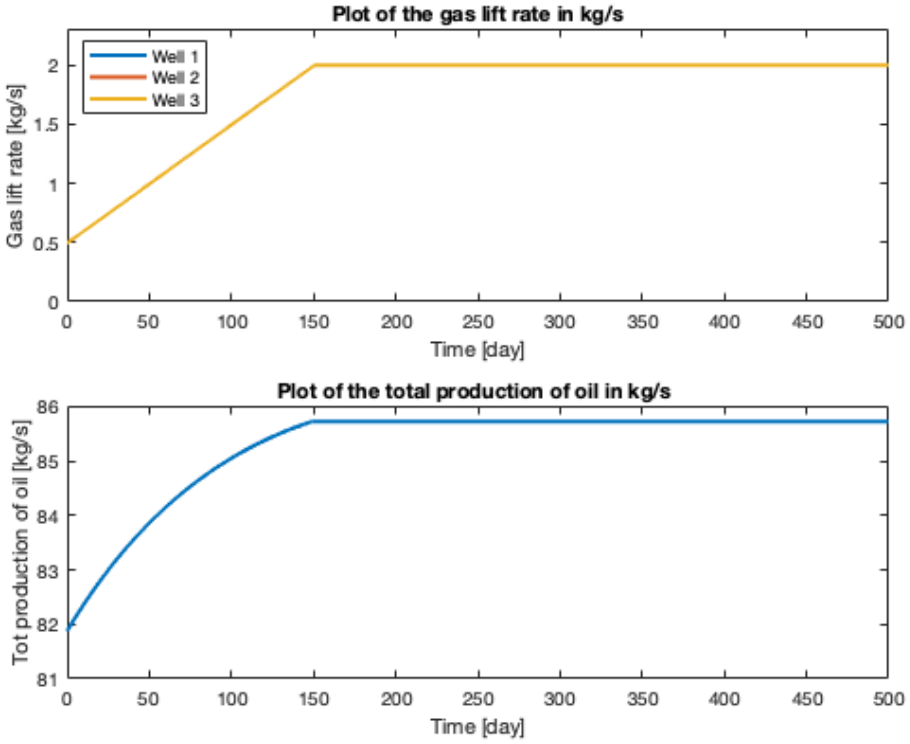

Figure 4.7: Result of the MPC with varying sand production rate- (initialized with sandproduction rate of 0.01 at the start)

Generally, an increase in levels of erosion was observed with increase in the gas-lift rate for the three wells. This is due to the increase in the pressure of the annulus and mass flow rate through the production valve when the gas-lift rate is increased. The greater the mass flow rate through the production valve, the greater the impact velocity of the sand particles hitting the surfaces of the choke valves, causing more erosion on the valves. Moreover, the effect of the well specific parameter GOR, gas-oil ratio on the erosion of the three wells was observed. The extent of erosion was noticed to increase with increase in the GOR inside the wells, as such well 2 was the well which was exposed to the most erosion during the simulation.

5 Conclusion and Recommendation

In this project, a combined control structure of diagnostics, prognostics and production optimization was proposed. The information acquired from the diagnostic module allows the controller to modify the control objectives such that it includes the system equipment health. In this manner, the controller generates control actions to satisfy both the control objectives, which is to maximize the total production and extend the useful life of critical components in the system. This combined control structure was applied to a gas-lifted subsea oil and gas production network subject to sand caused choke erosion.

It was showed that by combining a diagnostic prognostic module for the choke erosion to the control structure, the total production of oil can be maximized, while keeping the erosion levels of the choke valves under a certain threshold. Furthermore, the results with the different case studies shows that the controller manages to successfully keep the erosion of the three wells below the set threshold. However, more work is still needed to improve the performance of the MPC. Longer prediction horizon may improve the performance, as it will enable the controller to better adjust the gas-lift rates to keep the erosion below set threshold. In addition, high quality of measurement data is also required to provide better results from the trained models.

The objectives of this project was to showcase the performance of a control structure that incorporates equipment diagnostics. Data-driven models were used for the erosion diagnostics instead of the phenomenological erosion model. Regression modelling provided strong results in predicting the erosion on simulated data. Assuming, the simulations are a good representation of the erosion of a choke valve, the regression model appears to show a strong potential. There is, however, still a lot of further work to be done on this topic. The performance of other data-driven models such as auto-regressive exogenous input(ARX) and auto-regressive moving average exogenous input(ARMAX) should also be investigated to further validate that data driven models can be used in MPC.

References

[1] Dale E Seborg, Duncan A Mellichamp, Thomas F Edgar, and Francis J Doyle III. *Process dynamics and control*. John Wiley & Sons, 2010.

[2] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer-Verlag London Limited 1999, 1999.

[3] John D Hedengren, Reza Asgharzadeh Shishavan, Kody M Powell, and Thomas F Edgar. Nonlinear modeling, estimation and predictive control in apmonitor. *Computers & Chemical Engineering*, 70:133–148, 2014.

[4] A. Verheyleweghen and J. Jäschke. Oil production optimization of several wells subject to choke degradation. *IFAC-PapersOnLine*, 51:1–6, 2018.

[5] R Winston Revie. *Corrosion and corrosion control: an introduction to corrosion science and engineering*. John Wiley & Sons, 2008.

[6] Constantinos C Pantelides and JG Renfro. The online use of first-principles models in process operations: Review, current status and future needs. *Computers & chemical engineering*, 51:136–148, 2013.

[7] T Escobet, V Puig, and F Nejjari. Health aware control and model-based prognosis. In *2012 20th Mediterranean Conference on Control & Automation (MED)*, pages 691–696. IEEE, 2012.

[8] Hector Sanchez, Teresa Escobet, Vicenç Puig, and Peter Fogh Odgaard. Health-aware model predictive control of wind turbines using fatigue prognosis. *IFAC-PapersOnLine*, 48(21):1363–1368, 2015.

[9] Adriaen Verheyleweghen, Julie Marie Gjøby, and Johannes Jäschke. Health-aware operation of a subsea compression system subject to degradation. In *Computer Aided Chemical Engineering*, volume 43, pages 1021–1026. Elsevier, 2018.

[10] Raúl Hernández-Molinar, Roberto Sarmiento-Rebeles, and César F Méndez-Barrios. Least squares method and empirical modeling: A case study in a mexican manufacturing firm. *Empirical Modeling and Its Applications*, page 43, 2016.

[11] R Lyman Ott and Micheal T Longnecker. *An introduction to statistical methods and data analysis*. Nelson Education, 2015.

[12] Stephanie Glen. "Stepwise Regression". `https://www.statisticshowto.com/stepwise-regression/`, September 24, 2015. [Online; accessed 23-December-2020].

[13] Dinesh Krishnamoorthy, Bjarne Foss, and Sigurd Skogestad. Real-time optimization under uncertainty applied to a gas lifted well network. *Processes*, 4(4):52, 2016.

[14] GL DNV. Managing sand production and erosion. *Recommended Practice DNVGL-RP-O501. DNV GL Company, Oslo, Norway*, 2015.

[15] Joel AE Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

Appendices

:

1 Parameters for Erosion modelling

Table 1.1: Parameters used for erosion calculation

Parameter	Unit	Description	value
ρ_p	kg m^3	Density of sand particles	$2.5 \cdot 10^{-3}$
C_1	-	Model geometry factor	1.25
C_{unit}	mm m^{-1}	Unit conversion factor	1000
D	m	Length from cage and choke body	0.1
d_p	m	Sand particle diameter	$2.5 \cdot 10^{-4}$
GF	-	Geometry factor	2.0
H	m	Height of Gallery	0.3
K	-	Material erosion constant	$2 \cdot 10^{-9}$
Mm_g	$gmol^{-1}$	Molar mass of gas	20
$\dot{m}_p$	kgs^{-1}	Sand rate	$50 \cdot 10^{-2}$
n	-	Velocity exponent	2.6
r	m	Radius of curvature	0.2

2 Parameters for Gas-lift

Table 2.1: Parameters used in the gas lift model

Parameter	Unit	Description	value
μ_o	Pa s	Dynamic viscosity of oil	0.001
ρ_o	kgm^{-3}	Density of oil	$8 \cdot 10^2$
ρ_{ro}	kg m^{-3}	Density of oil in riser	$8 \cdot 10^2$
A_r	m^2	Cross-sectional area of riser	0.0115
C_{pr}	-	Valve constant for riser valve	0.01
D_r	m	Diameter of riser	0.121
H_r	m	Height of riser	500
L_r	m	Length of riser	500
n_w	-	Number of wells	3
p_s	bar	separator pressure	20
T	s	Sampling time	86400
T_r	K	Riser temperature	303

3 Well specific parameters in the gas lift model

Table 3.1: Well specified parameters used in the gas lift model

Parameter	Unit	Description	well 1	well 2	well 3
A_{bh}	m^2	Cross- sectional area of well below injection point	0.0115	0.0115	0.0115
A_w	m^2	Cross-sectional area of well above injection point	0.0115	0.0115	0.0115
C_{iv}	-	Valve constant for injection valve	0.0003	0.0003	0.0003
C_{pc}	-	Valve constant for production valve	0.002	0.002	0.002
D_a	m	Diameter of annulus	0.189	0.189	0.189
D_{bh}	m	Diameter of well below injection point	0.121	0.121	0.121
D_w	m	Diameter of well above injection point	0.121	0.121	0.121
GOR	-	Gas oil ration	0.10	0.12	0.11
H_a	m	Height of annulus	1000	1000	1000
H_{bh}	m	Height of tubing below injection point	500	500	500
H_w	m	Height of tubing above injection point	1000	1000	1000
L_a	m	Length of annulus	1500	1500	1500
L_{bh}	m	Length of pipe below injection point	500	500	500
L_w	m	Length of pipe above injection point	1500	1500	1500
p_r	bar	Reservoir pressure	150	155	160
PI	-	Reservoir productivity index	5	5	5
T_a	K	Ammullus temperature	301	301	301
T_w	K	Well temperature	305	305	305

4 List of Symbols

Table 4.1: List of symbols

Symbol	Unit	Description
α	rad	Characteristic impact angle
Δu_{max}	kg s^{-1}	Maximum change in input
$\dot{m}_p$	kg s^{-1}	Sand rate
γ	-	Relationship b/n particle diameter and diameter of choke
γ_c	-	Relative critical particle diameter
μ	kg $m^{-1}s^{-1}$	Dynamic viscosity
ρ	kg m^{-3}	Density
ρ_a	kg m^{-3}	Density of the gas in the annulus
ρ_o	kg m^{-3}	Density of the oil
ρ_p	kg m^{-3}	Density of sand particles
ρ_s	-	Weighting for slack variable
ρ_w	kg m^{-3}	Density of the mixture in the tubing
A	-	Dimensionless constant
A_g	m^2	Area of the the annulus
A_g	m^2	Effective gallery area
A_p	m^2	Area of pipe
A_r	m^2	Cross-sectional area of piping over the injection point
A_t	m^2	Area exposed to erosion
A_w	m^2	Cross-sectional area of piping under the injection point
C_1	-	Model geometry factor
C_{iv}	-	Valve constant for the injection valve
C_{pc}	-	Valve constant for the production valve
C_{unit}	mm m^{-1}	Unit conversion factor
D	m	Length from cage and choke body
d_p	m	Sand particle diameter
E	mm	Erosion
ER	mm yr^{-1}	Erosion rate
G	-	Particle size correction factor
g	m s^{-2}	Gravitational constant

5 Code for calculations

.

Listing 1: Code for calculating the parameters and the erosion using the well plant model

```
function [xk,zk] = WellPlantModel(dx0,z0,u0,par)

    addpath('/Users/SALI/Desktop/MATLAB/casadi-osx-matlabR2015a-v3.5.5')
    import casadi.*



    %% Parameters
    %number of wells
    n_w = par.n_w; %[]
    %gas constant
    R = par.R; %[m3 Pa /K /mol]
    %molecular weigth
    Mw = par.Mw; %[kg/mol?]

    %properties
    %density of oil - dim:  nwells x 1
    rho_o = par.rho_o; %[kg/m3]
    %riser oil density
    rho_ro = par.rho_ro;%[kg/m3]
    %1cP oil viscosity
    mu_oil = par.mu_oil;% [Pa s]

    %project
    %well parameters - dim:  nwells x 1
    L_w = par.L_w; %[m]
    H_w = par.H_w; %[m]
    D_w = par.D_w; %[m]
    A_w = par.A_w;%[m2]

    %well below injection - [m]
    L_bh = par.L_bh;
    H_bh = par.H_bh;
    D_bh = par.D_bh;
    A_bh = par.A_bh;%[m2]

    %annulus - [m]
    H_a = par.H_a;
    V_a = par.V_a; %[m3]

    %riser - [m]
    L_r = par.L_r;
    H_r = par.H_r;
    D_r = par.D_r;
    A_r = par.A_r;%[m2]

    %injection valve characteristics  - dim:  nwells x 1
    C_iv = par.C_iv;%[m2]
    %production valve characteristics  - dim:  nwells x 1
    C_pc = par.C_pc;%[m2]
    %riser valve characteristics
```

```
C_pr = par.C_pr;%[m2]
% account for differences in the vapor and oil velocity
slip = par.slip_real;


%% For erosion model
% Sand
d_p = par.d_p; %[m] particle diameter
rho_p = par.rho_p; %[kg/m3] particle density
mdot_p = par.mdot_p; %[kg/s] sand rate

% Choke
K = par.K; %[-] material erosion constant
rho_t = par.rho_t; %[kg/m3] sensity CS
r = par.r; %[m] radius of curvature
D = par.D; %[m] Gap between body and cage
H = par.H; %[m] Height of gallery

% Constants
C_unit = par.C_unit; % Unit conversion factor: now in mm/s
C_1 = par.C_1; %[-] Model/geometry factor
n = par.n; %[-] Velocity coefficient
GF = par.GF; %[-] Geometry factor

% Precalculations of erosion in choke:
alpha = par.alpha;
F = par.F;
A_g = par.A_g; %[m2] Effective gallery area
G = 1; % THIS MUST BE CHANGED
ER_constant = par.ER_constant;

gma = d_p./D;


%% Differential states
%symbolic declaration
%gas holdup @ annulus
m_ga = MX.sym('m_ga',n_w); % 1-2 [ton]
%gas holdup @ well
m_gt = MX.sym('m_gt',n_w); % 3-4 [ton]
%oil holdup @ well
m_ot = MX.sym('m_ot',n_w); % 5-6 [ton]
%gas holdup @ riser
m_gr = MX.sym('m_gr',1);   % 7 [ton]
%oil holdup @ riser
m_or = MX.sym('m_or',1);   % 8 [ton]

%% Algebraic states
%pressure - annulus
p_ai = MX.sym('p_ai',n_w);      % 1-2 [bar] (bar to Pa  = x10^5)
%pressure - well head
p_wh = MX.sym('p_wh',n_w);      % 3-4 [bar]
%pressure - injection point
p_wi = MX.sym('p_wi',n_w);      % 5-6 [bar]
%pressure - below injection point (bottom hole)
p_bh = MX.sym('p_bh',n_w);      % 7-8 [bar]
%density - annulus
rho_ai = MX.sym('rho_ai',n_w);  % 9-10 [100 kg/m3]
```

```
%mixture density in tubing
rho_m = MX.sym('rho_m',n_w);     % 11-12 [100 kg/m3]
%well injection flow rate
w_iv = MX.sym('w_iv',n_w);       % 13-14 [kg/s]
%wellhead total production rate
w_pc = MX.sym('w_pc',n_w);       % 15-16 [kg/s]
%wellhead gas production rate
w_pg = MX.sym('w_pg',n_w);       % 17-18 [kg/s]
%wellhead oil production rate
w_po = MX.sym('w_po',n_w);       % 19-20 [kg/s]
%oil rate from reservoir
w_ro = MX.sym('w_ro',n_w);       % 21-22 [kg/s]
%gas rate from reservoir
w_rg = MX.sym('w_rg',n_w);       % 23-24 [0.1 kg/s]
%riser head pressure
p_rh = MX.sym('p_rh',1);         % 25 [bar]
%mixture density in riser
rho_r =  MX.sym('rho_r',1);      % 26 [100 kg/s]
%manifold pressure
p_m = MX.sym('p_m',1);           % 27 [bar]
%riser head total production rate
w_pr = MX.sym('w_pr',1);         % 28 [kg/s]
%riser head total oil production rate
w_to = MX.sym('w_to',1);         % 29 [kg/s]
%riser head total gas production rate
w_tg = MX.sym('w_tg',1);         % 30 [kg/s]

%control input
%gas lift rate
w_gl = MX.sym('w_gl',n_w); %[kg/s]

%parameters
p_res = MX.sym('p_res',n_w);
%productivity index
PI = MX.sym('PI',n_w); %[kg s^-1 bar-1]
%GasOil ratio
GOR = MX.sym('GOR',n_w); %[kg/kg]
%Annulus temperature
T_a = MX.sym('T_a',n_w); %[oC]
%well temperature
T_w = MX.sym('T_w',n_w); %[oC]
%riser temperature
T_r = MX.sym('T_r',1); %[oC]
%separator pressure
p_s = MX.sym('p_s',1); %[bar]
%time transformation: CASADI integrates always from 0 to 1 and
%the USER does the time
%scaling with T.
T = MX.sym('T',1); %[s]
%erosion rate
ER = MX.sym('ER',n_w); %[s]
%g1
g1 = MX.sym('g1',n_w);
%mixed dynamic viscosity
mu_f = MX.sym('mu_f',n_w);
%particle impact velocity
V_p = MX.sym('V_p',n_w);
```

```
    %% Modeling
    %gas fraction (mass) of the well holdup - avoiding zero division
    xGwH = (m_gt.*1e3./max(1e-3,(m_gt.*1e3+m_ot.*1e3)));
    %gas fraction (mass) of the riser holdup
    xGrH = (m_gr.*1e3./(m_gr.*1e3+m_or.*1e3));
    xGw = slip.*xGwH./(1 + (slip-1).*xGwH);
    xOw = 1 - xGw;
    xGr = slip.*xGrH./(1 + (slip-1).*xGrH);
    xOr = 1 - xGr;

    % ==================================
    %     Well model with/withou pressure loss
    % ==================================
    % algebraic equations (all symbolic)
    %annulus pressure - %g = 9.81
    f1 = -p_ai.*1e5 + ((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;
    %well head pressure
    f2 = -p_wh.*1e5 + ((R.*T_w./Mw).*(m_gt.*1e3./(L_w.*A_w + L_bh.*A_bh...
    - m_ot.*1e3./rho_o))) - ((m_gt.*1e3+m_ot.*1e3 )./(L_w.*A_w)).*9.81...
.*H_w/2;
    %well injection point pressure
    f3 = -p_wi.*1e5 + (p_wh.*1e5 + 9.81./(A_w.*L_w).*max(0,(m_ot.*1e3...
    +m_gt.*1e3-rho_o.*L_bh.*A_bh)).*H_w) + (128.*mu_oil.*L_w.*w_pc./...
(3.14.*D_w.^4.*((m_gt.*1e3 + m_ot.*1e3).*p_wh.*1e5.*Mw.*rho_o)./...
(m_ot.*1e3.*p_wh.*1e5.*Mw + rho_o.*R.*T_w.*m_gt.*1e3)));
    %bottom hole pressure
    f4 = -p_bh.*1e5 + (p_wi.*1e5 + rho_o.*9.81.*H_bh + 128.*mu_oil...
    .*L_bh.*w_ro./(3.14.*D_bh.^4.*rho_o));
    %gas density in annulus
    f5 = -rho_ai.*1e2 +(Mw./(R.*T_a).*p_ai.*1e5);
    %fluid mixture density in well
    f6 = -rho_m.*1e2 + ((m_gt.*1e3 + m_ot.*1e3).*p_wh.*1e5.*Mw.*rho_o)./...
    (m_ot.*1e3.*p_wh.*1e5.*Mw + rho_o.*R.*T_w.*m_gt.*1e3);
    %well injection flow rate
    f7 = -w_iv + C_iv.*sqrt(rho_ai.*1e2.*max(0,(p_ai.*1e5 - p_wi.*1e5)));
    %wellhead prodution rate
    f8 = -w_pc + 1.*C_pc.*sqrt(rho_m.*1e2.*max(0,(p_wh.*1e5 - p_m.*1e5)));
    %wellhead gas production rate
    f9 = -w_pg + xGw.*w_pc;
    %wellhead oil prodution rate
    f10 = -w_po + xOw.*w_pc;
    %oil from reservoir flowrate
    f11 = -w_ro + PI.*1e-6.*(p_res.*1e5 - p_bh.*1e5);
    %gas from reservoir production rate
    f12 = -w_rg.*1e-1 + GOR.*w_ro;
    %riser head pressure
    f13 = -p_rh.*1e5 + ((R.*T_r./Mw).*(m_gr.*1e3./(L_r.*A_r))) - ...
    ((m_gr.*1e3+m_or.*1e3 )./(L_r.*A_r)).*9.81.*H_r/2;
    %riser density
    f14 = -rho_r.*1e2 + ((m_gr.*1e3 + m_or.*1e3).*p_rh.*1e5.*Mw.*...
    rho_ro)./(m_or.*1e3.*p_rh.*1e5.*Mw + rho_ro.*R.*T_r.*m_gr.*1e3);
    %manifold pressure
    f15 = -p_m.*1e5 + (p_rh.*1e5 + 9.81./(A_r.*L_r).*(m_or.*1e3+m_gr.*...
    1e3).*H_r) + (128.*mu_oil.*L_r.*w_pr./(3.14.*D_r.^4.*((m_gr.*1e3 +...
m_or.*1e3).*p_rh.*1e5.*Mw.*rho_ro)./(m_or.*1e3.*p_rh.*1e5.*Mw + ...
rho_ro.*R.*T_r.*m_gr.*1e3)));
```

```
%total production rate of well
f16 = -w_pr + 1.*C_pr.*sqrt(rho_r.*1e2.*(p_rh.*1e5 - p_s.*1e5));
%oil total production rate
f17 = -w_to + xOr.*w_pr;
%gas total production rate
f18 = -w_tg + xGr.*w_pr;
% setting differential equations as algebraic equations since the
% dynamics of ER is on a much larger time scale
f19 = (w_gl - w_iv).*1e-3;
f20 = (w_iv + w_rg.*1e-1 - w_pg).*1e-3;
f21 = (w_ro - w_po).*1e-3;
f22 = (sum(w_pg) - w_tg).*1e-3 ;
f23 = (sum(w_po) - w_to).*1e-3 ;
f24 = - V_p + 3/(4*A_g)*(w_po/rho_o + R*T_w.*w_pg./(p_wh.*10^5*Mw));
f25 = - mu_f + mu_oil.*(w_po/rho_o)./(w_po./rho_o + R.*T_w.*w_pg./...
(p_wh.*10^5*Mw));
% Assuming that gamma < 0 (checked in main)
f26 = -g1 + gma/0.1;

% differential equations - (all symbolic) - [ton]
% Erosion rate
df1 =  ER_constant.*g1.*(V_p).^n;

% Form the DAE system
diff = vertcat(df1);
alg = vertcat(f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,...
f15,f16,f17,f18,f19,f20,f21,f22,f23,f24,f25,f26);

% give parameter values
alg = substitute(alg,p_res,par.p_res);
alg = substitute(alg,p_s,par.p_s);
alg = substitute(alg,T_a,par.T_a);
alg = substitute(alg,T_w,par.T_w);
alg = substitute(alg,T_r,par.T_r);

diff = substitute(diff,p_res,par.p_res);
diff = substitute(diff,T_w,par.T_w);

% concatenate the differential and algebraic states
x_var = vertcat(ER);
z_var = vertcat(p_ai,p_wh,p_wi,p_bh,rho_ai,rho_m,w_iv,w_pc,...
w_pg,w_po,w_ro,w_rg,p_rh,rho_r,p_m,...
    w_pr,w_to,w_tg,m_ga,m_gt,m_ot,m_gr,m_or,V_p, mu_f,g1);
p_var = vertcat(w_gl,GOR,PI,T);

%end modeling
%% Casadi commands
%declaring function in standard DAE form (scaled time)
dae = struct('x',x_var,'z',z_var,'p',p_var,'ode',T*diff,'alg',alg);

%calling the integrator, the necessary inputs are: label;
%integrator; function with IO scheme of a DAE (formalized); struct
%(options)
F = integrator('F','idas',dae);

%assuming inputs as symbolic in order to obtain the gradients
%symbolically
theta = MX.sym('theta',3);
```

```
    %integration results
    Fend = F('x0',dx0,'z0',z0,'p',[u0;par.GOR;par.PI;par.T]);
    %extracting the results (from symbolic to numerical)
    xk = full(Fend.xf);
    zk = full(Fend.zf);
    %xf = Fend.xf;
end
```

Listing 2: Code for generating training data

```
clear
close all
clc

%% noise --> For reproducibility
% This is choosing a seed for generating random numbers
rng(1)

%%

%% Initializing the table to store multiple time series:

nTimeseries = 1;
dataSz = [nTimeseries 10];
varNames = {'Num','uArray', 'yMeas','erosionArray', 'H', 'x0', 'xk',...
    'yk', 'z0', 'zk'};
varTypes = {'double', 'cell', 'cell', 'cell', 'cell', 'cell', 'cell',...
    'cell', 'cell', 'cell'};
Data = table('Size', dataSz, 'VariableNames', varNames, 'VariableTypes',...
varTypes);


%% Model initialization
sandArray = sandproductionrate(0.02,500,'exp',0.005);
%[sandArray,sandArrayNoise,stepSandArray] =
%sandproductionrate(0.01,500,'exp',0.015); will be deactivated all the time
for i_timeseries = 1:nTimeseries



% initial condition (pre-computed)
[x0,z0,u0] = InitialConditionGasLift_5;

% model parameters
%par = ParametersGasLift(1);
par = ParametersGasLift(1,sandArray);

%states to measurement mapping function
H = zeros(16,length(z0));
%pai - annulus pressure, well 1-3
H(1,1) = 1;
H(2,2) = 1;
H(3,3) = 1;
%pwh - well head pressure, well+ 1-3
H(4,4) = 1;
H(5,5) = 1;
```

```
H(6,6) = 1;
%wro - wellhead gas production rate, well 1-3
H(7,25) = 1;
H(8,26) = 1;
H(9,27) = 1;
%wrg - wellhead oil production rate, well 1-3
H(10,28) = 1;
H(11,29) = 1;
H(12,30) = 1;
%prh - riser head pressure
H(13,37) = 1;
%pm - manifold pressure
H(14,39) = 1;
%wto - riser head total oil production rate
H(15,41) = 1;
%wtg - riser head total gas production rate
H(16,42) = 1;



%% Simulation parameters

% Number of simulation steps
nSim = 500;  % time_total = 3600*24*500; %[s]



[dx0,z0,u0] = InitialConditionGasLift_5

%sampling time /control interval /1 simulation iteration time
par.T = 3600*24; % [s]



%% Simulation initialization
xk = x0;
zk = z0;
uk = u0;
yk = H*z0;


%%
fprintf('Time series number: >>> %0.0f \n',i_timeseries)
%creating random array for the inputs
uArray = [];
%bounds on the inputs
uMin = 1.5;
uMax = 2.5;

for t = 0:nSim
    if rem(t,50) == 0 %every 50 days we change the inputs
        uk = (uMax - uMin).*rand(3,1) + uMin;
    end
    uArray = [uArray, uk];
end
```

```
% measurements (for plotting)
yMeas = yk;
erosionArray = xk;


for u = 1:nSim

    %fprintf('     iteration >>> %0.0f \n',t)

    % integrating the system
    [xk,zk] = WellPlantModel(xk,zk,uArray(:,u),par);
    par = ParametersGasLift(u,sandArray);
    par.T = 3600*24;
    % Adding noise to the measurements
    yMeas = [yMeas, H*zk + par.scale.*randn(length(yk),1)];
    erosionArray = [erosionArray, xk];
end


Data(i_timeseries, :) = {i_timeseries, uArray, yMeas, erosionArray,...
    H, x0, xk, yk, z0, zk};

end

%% saving the data in a mat file
filename = 'datamatrix_'+string(nTimeseries)+'.mat';
save(filename, 'Data')
%% saving the matrix in a mat file
%filename1 = 'hmatrix' + '.mat';
%save(filename1,'H')

%% Plotting
figure(1)


time = 0:1:500; %[days]


% System inputs
subplot(2,1,1)
    stairs(time,uArray(1,:),'LineWidth',2);
    hold on
    stairs(time,uArray(2,:),'LineWidth',2);
    stairs(time,uArray(3,:),'LineWidth',2);
    legend('Well 1','Well 2','Well 3');

    ylim([0,5]);
    xlabel('Time [day]');
    ylabel('Gas lift rate [kg/s]');

% erosion
subplot(2,1,2);
    plot(time,transpose(Data.erosionArray{1,1}(1,:)),'LineWidth',2);
    hold on
    plot(time,transpose(Data.erosionArray{1,1}(2,:)),'LineWidth',2);
    plot(time,transpose(Data.erosionArray{1,1}(3,:)),'LineWidth',2);
    %ylim([0,5.2]);
    legend('Well 1','Well 2','Well 3');
```

```
    legend('Location','northwest');
    xlabel('Time [day]');
    ylabel('Erosion [mm]');

figure(2)

% Pressure
subplot(2,1,1)
    plot(time,yMeas(4,:),'LineWidth',1.5);
    hold on
    plot(time,yMeas(5,:),'LineWidth',1.5);
    plot(time,yMeas(6,:),'LineWidth',1.5);
    axis([0 500 45 60]);
    legend('Well 1','Well 2','Well 3');

    %ylim([0,2.2]);
    xlabel('Time [day]');
    ylabel('Well head pressure [bar]');

% erosion
subplot(2,1,2);
    plot(time,yMeas(15,:),'LineWidth',2); %oil
    %plot(time,yMeas(14,:)); %gas
    axis([0 500 50 100]);

    xlabel('Time [day]');
    ylabel('Flowrate [kg/s]');
```

Listing 3: Code for the controller

```
function [u_,s,w_,exitflag] = NMPC(x_current,ui,z_current,nm,np...
                                                    ,n,sandArray)

addpath('/Users/SALI/Desktop/MATLAB/casadi-osx-matlabR2015a-v3.5.5')
import casadi.*

nu = 3;
nx = 3;
nz = 62;

%% Parameters
par = ParametersGasLift(n,sandArray);   %For varying sand production rate
%par = ParametersGasLift;                % For constant sand production
%number of wells
n_w = 3; %[]
%gas constant
R = par.R; %[m3 Pa /K /mol]
%molecular weigth
Mw = par.Mw; %[kg/mol]

%properties
%density of oil - dim:  nwells x 1
rho_o = par.rho_o; %[kg/m3]
%riser oil density
rho_ro = par.rho_ro;%[kg/m3]
%1cP oil viscosity
mu_oil = par.mu_oil;% [Pa s]
```

```
%project
%well parameters - dim:  nwells x 1
L_w = par.L_w; %[m]
H_w = par.H_w; %[m]
D_w = par.D_w; %[m]
A_w = par.A_w;%[m2]

%well below injection - [m]
L_bh = par.L_bh;
H_bh = par.H_bh;
D_bh = par.D_bh;
A_bh = par.A_bh;%[m2]

%annulus - [m]
H_a = par.H_a;
V_a = par.V_a; %[m3]

%riser - [m]
L_r = par.L_r;
H_r = par.H_r;
D_r = par.D_r;
A_r = par.A_r;%[m2]

%injection valve characteristics  - dim:  nwells x 1
C_iv = par.C_iv;%[m2]
%production valve characteristics  - dim:  nwells x 1
C_pc = par.C_pc;%[m2]
%riser valve characteristics
C_pr = par.C_pr;%[m2]
% account for differences in the vapor and oil velocity
slip = par.slip_real;


%% For erosion model
% Sand
d_p = par.d_p; %[m] particle diameter
rho_p = par.rho_p; %[kg/m3] particle density
mdot_p = par.mdot_p; %[kg/s] sand rate

% Choke
K = par.K; %[-] material erosion constant
rho_t = par.rho_t; %[kg/m3] sensity CS
r = par.r; %[m] radius of curvature
D = par.D; %[m] Gap between body and cage
H = par.H; %[m] Height of gallery

% Constants
C_unit = par.C_unit; % Unit conversion factor: now in mm/s
C_1 = par.C_1; %[-] Model/geometry factor
n = par.n; %[-] Velocity coefficient
GF = par.GF; %[-] Geometry factor

% Precalculations of erosion in choke:
alpha = par.alpha;
F = par.F;
A_g = par.A_g; %[m2] Effective gallery area
ER_constant = par.ER_constant;
```

```
gma = d_p./D;


%% Algebraic states
%pressure - annulus
p_ai = MX.sym('p_ai',n_w);      % 1-3 [bar] (bar to Pa  = x10^5)
%pressure - well head
p_wh = MX.sym('p_wh',n_w);      % 4-6 [bar]
%pressure - injection point
p_wi = MX.sym('p_wi',n_w);      % 7-9 [bar]
%pressure - below injection point (bottom hole)
p_bh = MX.sym('p_bh',n_w);      % 10-12 [bar]
%density - annulus
rho_ai = MX.sym('rho_ai',n_w);  % 13-15 [100 kg/m3]
%mixture density in tubing
rho_m = MX.sym('rho_m',n_w);    % 16-18 [100 kg/m3]
%well injection flow rate
w_iv = MX.sym('w_iv',n_w);      % 19-21 [kg/s]
%wellhead total production rate
w_pc = MX.sym('w_pc',n_w);      % 22-24 [kg/s]
%wellhead gas production rate
w_pg = MX.sym('w_pg',n_w);      % 25-27 [kg/s]
%wellhead oil production rate
w_po = MX.sym('w_po',n_w);      % 28-30 [kg/s]
%oil rate from reservoir
w_ro = MX.sym('w_ro',n_w);      % 31-33 [kg/s]
%gas rate from reservoir
w_rg = MX.sym('w_rg',n_w);      % 34-36 [0.1 kg/s]
%riser head pressure
p_rh = MX.sym('p_rh',1);        % 37 [bar]
%mixture density in riser
rho_r =  MX.sym('rho_r',1);     % 38 [100 kg/s]
%manifold pressure
p_m = MX.sym('p_m',1);          % 39 [bar]
%riser head total production rate
w_pr = MX.sym('w_pr',1);        % 40 [kg/s]
%riser head total oil production rate
w_to = MX.sym('w_to',1);        % 41 [kg/s]
%riser head total gas production rate
w_tg = MX.sym('w_tg',1);        % 42 [kg/s]
%gas holdup @ annulus
m_ga = MX.sym('m_ga',n_w); % 43-45 [ton]
%gas holdup @ well
m_gt = MX.sym('m_gt',n_w); % 46-48 [ton]
%oil holdup @ well
m_ot = MX.sym('m_ot',n_w); % 49-51 [ton]
%gas holdup @ riser
m_gr = MX.sym('m_gr',1);   % 52 [ton]
%oil holdup @ riser
m_or = MX.sym('m_or',1);   % 53 [ton]
%particle impact velocity
V_p = MX.sym('V_p',n_w); % 54-56
%dynamic viscosity of mixture
mu_f = MX.sym('mu_f',n_w); % 57-59
g1 = MX.sym('g1',n_w); % 60-62


%control input
```

```
%gas lift rate
w_gl = MX.sym('w_gl',n_w); %[kg/s]

%parameters
p_res = MX.sym('p_res',n_w);
%productivity index
PI = MX.sym('PI',n_w); %[kg s^-1 bar-1]
%GasOil ratio
GOR = MX.sym('GOR',n_w); %[kg/kg]
%Annulus temperature
T_a = MX.sym('T_a',n_w); %[oC]
%well temperature
T_w = MX.sym('T_w',n_w); %[oC]
%riser temperature
T_r = MX.sym('T_r',1); %[oC]
%separator pressure
p_s = MX.sym('p_s',1); %[bar]
%time transformation: CASADI integrates always from 0 to 1 and the
%                                                 USER does the time
%scaling with T.
T = MX.sym('T',1); %[s]
%erosion rate
ER = MX.sym('ER',n_w); %

%% Modeling
%gas fraction (mass) of the well holdup - avoiding zero division
xGwH = (m_gt.*1e3./max(1e-3,(m_gt.*1e3+m_ot.*1e3)));
%gas fraction (mass) of the riser holdup
xGrH = (m_gr.*1e3./(m_gr.*1e3+m_or.*1e3));

xGw = slip.*xGwH./(1 + (slip-1).*xGwH);
xOw = 1 - xGw;
xGr = slip.*xGrH./(1 + (slip-1).*xGrH);
xOr = 1 - xGr;

% ==================================
%     Well model with/withou pressure loss
% ==================================
% algebraic equations (all symbolic)
%annulus pressure - %g = 9.81
f1 = -p_ai.*1e5 + ((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;
%well head pressure
f2 = -p_wh.*1e5 + ((R.*T_w./Mw).*(m_gt.*1e3./(L_w.*A_w + L_bh.*A_bh - ...
m_ot.*1e3./rho_o))) - ((m_gt.*1e3+m_ot.*1e3 )./(L_w.*A_w)).*9.81.*H_w/2;
%well injection point pressure
f3 = -p_wi.*1e5 + (p_wh.*1e5 + 9.81./(A_w.*L_w).*max(0,(m_ot.*1e3+m_gt.*...
1e3-rho_o.*L_bh.*A_bh)).*H_w) + (128.*mu_oil.*L_w.*w_pc./(3.14.*D_w.^4.*...
((m_gt.*1e3 + m_ot.*1e3).*p_wh.*1e5.*Mw.*rho_o)./(m_ot.*1e3.*p_wh.*1e5.*...
Mw + rho_o.*R.*T_w.*m_gt.*1e3)));
%bottom hole pressure
f4 = -p_bh.*1e5 + (p_wi.*1e5 + rho_o.*9.81.*H_bh + 128.*mu_oil.*L_bh.*...
w_ro./(3.14.*D_bh.^4.*rho_o));
%gas density in annulus
f5 = -rho_ai.*1e2 +(Mw./(R.*T_a).*p_ai.*1e5);
%fluid mixture density in well
f6 = -rho_m.*1e2 + ((m_gt.*1e3 + m_ot.*1e3).*p_wh.*1e5.*Mw.*rho_o)./...
```

```
(m_ot.*1e3.*p_wh.*1e5.*Mw + rho_o.*R.*T_w.*m_gt.*1e3);
%well injection flow rate
f7 = -w_iv + C_iv.*sqrt(rho_ai.*1e2.*max(0,(p_ai.*1e5 - p_wi.*1e5)));
%wellhead prodution rate
f8 = -w_pc + 1.*C_pc.*sqrt(rho_m.*1e2.*max(0,(p_wh.*1e5 - p_m.*1e5)));
%wellhead gas production rate
f9 = -w_pg + xGw.*w_pc;
%wellhead oil prodution rate
f10 = -w_po + xOw.*w_pc;
%oil from reservoir flowrate
f11 = -w_ro + PI.*1e-6.*(p_res.*1e5 - p_bh.*1e5);
%gas from reservoir production rate
f12 = -w_rg.*1e-1 + GOR.*w_ro;
%riser head pressure
f13 = -p_rh.*1e5 + ((R.*T_r./Mw).*(m_gr.*1e3./(L_r.*A_r))) - ((m_gr.*1e3...
+m_or.*1e3 )./(L_r.*A_r)).*9.81.*H_r/2;
%riser density
f14 = -rho_r.*1e2 + ((m_gr.*1e3 + m_or.*1e3).*p_rh.*1e5.*Mw.*rho_ro)./...
(m_or.*1e3.*p_rh.*1e5.*Mw + rho_ro.*R.*T_r.*m_gr.*1e3);
%manifold pressure
f15 = -p_m.*1e5 + (p_rh.*1e5 + 9.81./(A_r.*L_r).*(m_or.*1e3+m_gr.*1e3)...
.*H_r) + (128.*mu_oil.*L_r.*w_pr./(3.14.*D_r.^4.*((m_gr.*1e3 + m_or.*1e3)...
.*p_rh.*1e5.*Mw.*rho_ro)./(m_or.*1e3.*p_rh.*1e5.*Mw + rho_ro.*R.*T_r.*...
m_gr.*1e3)));
%total production rate of well
f16 = -w_pr + 1.*C_pr.*sqrt(rho_r.*1e2.*(p_rh.*1e5 - p_s.*1e5));
%oil total production rate
f17 = -w_to + xOr.*w_pr;
%gas total production rate
f18 = -w_tg + xGr.*w_pr;
% setting differential equations as algebraic equations since the
% dynamics of ER is on a much larger time scale
f19 = (w_gl - w_iv).*1e-3;
f20 = (w_iv + w_rg.*1e-1 - w_pg).*1e-3;
f21 = (w_ro - w_po).*1e-3;
f22 = (sum(w_pg) - w_tg).*1e-3 ;
f23 = (sum(w_po) - w_to).*1e-3 ;
f24 = - V_p + 3/(4*A_g).*(w_po./rho_o + R*T_w.*w_pg./(p_wh.*10^5*Mw));
f25 = - mu_f + mu_oil.*(w_po./rho_o)./(w_po./rho_o + R.*T_w.*w_pg./...
(p_wh.*10^5*Mw));
f26 = -g1 + gma/0.1;
% differential equations - (all symbolic) - [ton]
% Erosion rate
df1 =  ER_constant.*g1.*(V_p).^n;

% Form the DAE system
diff = vertcat(df1);
alg = vertcat(f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15,f16,...
f17,f18,f19,f20,f21,f22,f23,f24,f25,f26);

% give parameter values
alg = substitute(alg,p_res,par.p_res);
alg = substitute(alg,p_s,par.p_s);
alg = substitute(alg,T_a,par.T_a);
alg = substitute(alg,T_w,par.T_w);
alg = substitute(alg,T_r,par.T_r);

diff = substitute(diff,p_res,par.p_res);
```

```
diff = substitute(diff,T_w,par.T_w);

% concatenate the differential and algebraic states
x_var = vertcat(ER);
z_var = vertcat(p_ai,p_wh,p_wi,p_bh,rho_ai,rho_m,w_iv,w_pc,w_pg,w_po...
,w_ro,w_rg,p_rh,rho_r,p_m,...
    w_pr,w_to,w_tg,m_ga,m_gt,m_ot,m_gr,m_or,V_p, mu_f,g1);
p_var = vertcat(w_gl,GOR,PI,T);

%% NMPC
umax = 2;
umin = 0.4;
dumax = 0.01;
x_threshold = 2;


%% Defining empty nlp-problem
J = 0;
w = {};
w0 = [];
lbw =[];
ubw = [];
lbg = [];
ubg = [];
g = {};

x_prev = MX.sym('X0',nx);
Z0 = MX.sym('Z0',nz);
w = {w{:},x_prev}; % 1-3
lbw = [lbw,zeros(nx,1)];
ubw = [ubw,inf*ones(nx,1)];
w0 = [w0;x_current];

g = {g{:},x_prev};
lbg = [lbg,x_current];
ubg = [ubg,x_current];


%% Lifting initial conditions
uk = MX.sym('uk_init',nu);
w = {w{:}, uk}; % 4-6
w0 = [w0;  0.3*ones(nu,1)];
lbw = [lbw;ui];
ubw = [ubw;ui];


% uprev= MX.sym('uprev_init',nu);
U0 = MX.sym('U0',nu);
p = MX.sym('p',7);

x = MX.sym('x',nx);
uprev1 = uk;
uk1 = uk;
rho = 999999;
s = MX.sym('s',nx);
R = [1,0,0
    0,1,0
    0,0,1];
```

```
L =  -sum(w_po)+ rho*sum(s)  + 1/2 * ((uk1-U0)'*R*(uk1-U0));
f = Function('f',{x_var,z_var,p_var,uk1,U0,s},{diff,alg,L});

%% Using 3 collocation points:

h = par.T;
% Radau
t = [collocation_points(3, 'radau')];

% Finding M
M = [t',0.5*t'.^2,1/3*t'.^3]*inv([[1;1;1],t',t'.^2]);

%% Looping through until timeend
for k = 1:np
    uprev = uk;
    uk = MX.sym(['uk_' num2str(k)],nu);
    w = {w{:}, uk}; % 7-9
    w0 = [w0;  ui];
    lbw = [lbw;umin*ones(nu,1)];
    ubw = [ubw;umax*ones(nu,1)];

    s = MX.sym(['s_',num2str(k)],nx);
    w = {w{:},s}; % 10-12
    lbw = [lbw;0*ones(nu,1)];
    ubw = [ubw;1*ones(nu,1)];
    w0 = [w0;0*ones(nu,1)];

    % Adding constraint for delta_u
    duk = uk - uprev;
    g = {g{:},duk};

    if k > nm
        lbg = [lbg;zeros(nu,1)];
        ubg = [ubg; zeros(nu,1)];
    else
        lbg = [lbg;-dumax*ones(nu,1)];
        ubg = [ubg;dumax*ones(nu,1)];
    end

    % Collocation points
    fk = [];
    Xk1 = [];
    zk = [];
    L1 = [];
    for d = 1:3
        Xk = MX.sym(['Xk_' num2str(k),'_',num2str(d)],nx);
        Zk = MX.sym(['Zk_' num2str(k),'_',num2str(d)],nz);
        Xk1 = [Xk1,Xk];
        w = {w{:}, Xk, Zk}; % 13-15
        w0 = [w0;x_current;z_current];
        lbw = [lbw;zeros(nx,1);0*ones(nz,1)];
        ubw = [ubw;inf*ones(nx,1);inf*ones(nz,1)];

        % Calculating xdot and objective function
        [fk1,zk1,L] = f(Xk,Zk,vertcat(uk,p),uk,uprev,s);
        L1 = [L1;L];
        if k > nm
            L1(d) = L - 1/2 * ((uk-uprev)'*R*(uk-uprev));
```

```
        end
        fk = [fk, fk1];
        zk = [zk,zk1];
    end
    ML = M*L1;
    J = J + ML(3);
%         ML = M(3,1)*sum(zk(28:30,1)) + M(3,2)*sum(zk(28:30,2))
                    %+ M(3,3)*sum(zk(28:30,3)) +rho*sum(s);
%         J = J + ML;
    x_next1 = [];
    for d = 1:3
        % Calculating M*xdot for each collocation point
        Mfk = M(d,1)*fk(:,1) + M(d,2)*fk(:,2) + M(d,3)*fk(:,3);
        % Calculating x
        x_next = x_prev+h*Mfk;
        x_next1 = [x_next1,x_next];
        % Adding xk and Xk1 as constrains as they must be equal.
        g = {g{:},x_next-Xk1(:,d),zk(:,d)};
        lbg = [lbg;zeros(nx,1);zeros(nz,1)];
        ubg = [ubg;zeros(nx,1);zeros(nz,1)];
    end

    % New NLP variable for state at end
    x_prev = MX.sym(['x_init_' num2str(k)],nx);
    w = {w{:}, x_prev};
    w0 = [w0;x_current];
    lbw = [lbw;zeros(nx,1)];
    ubw = [ubw;inf*ones(nx,1)];

    % Gap
    g = {g{:},x_next-x_prev};
    lbg = [lbg;zeros(nx,1)];
    ubg = [ubg;zeros(nx,1)];

    % Constraint on erosion
    g = {g{:},x_prev-s};
    lbg = [lbg;zeros(nx,1)];
    ubg = [ubg;x_threshold*ones(nx,1)];

end

%% Solving optimization problem

% Formalizing problem
nlp = struct('x',vertcat(w{:}),'g',vertcat(g{:}),'f',J,'p',vertcat(U0,p));

% Assigning solver (IPOPT)
solver = nlpsol('solver','ipopt',nlp);

% Solving the problem
sol = solver('x0',w0,'lbx',lbw,'ubx',ubw,'lbg',lbg,'ubg',ubg,'p',...
[ui;par.GOR;par.PI;par.T]);

%% Extracting solution
w_opt = full(sol.x);
obj_opt = -full(sol.f);
c_opt = full(sol.g);
lag_opt = full(sol.lam_g);
```

```
u_ = w_opt(7:9);
s = w_opt(10:12);
w_ = w_opt(43:45);
exitflag = solver.stats.success;
end
```

Listing 4: Code for initial conditions used in the calculation

```
function [dx0,z0,u0] = InitialConditionGasLift_5

%% Differential states

%well erosion rate
ER0 = [1,1,1]'/(365*24*3600); %[mm/s] 9-11  seconds in a year

%% Algebraic states
%pressure - annulus
p_ai0 = [61.9230, 62.1454, 62]';%[bar] 1-3  (bar to Pa  = x10^5)
%pressure - well head
p_wh0 = [42.5851, 45.3082, 44]';%[bar] 4-6
%pressure - injection point
p_wi0 = [56.8713, 57.1119, 57]';%[bar] 7-9
%pressure - below injection point (bottom hole)
p_bh0 = [96.1433, 98.3867, 96.25]';%[bar] 10-12
%density - annulus
rho_ai0 = [0.4949, 0.4967, 0.4955]';%[100 kg/m3] 13-15
%mixture density in tubing
rho_m0 = [2.3400, 2.2359, 2.3]';%[100 kg/m3] 16-18
%well injection flow rate
w_iv0 = [0.5000, 0.5000,0.5000]';%[kg/s] 19-21
%wellhead total production rate
w_pc0 = [30.1212, 33.3235,32]';%[kg/s] 22-24
%wellhead gas production rate
w_pg0 = [3.1928, 4.0168, 4]';%[kg/s] 25-27
%wellhead oil production rate
w_po0 = [26.9283, 29.3067, 28]';%[kg/s] 28-30
%oil rate from reservoir
w_ro0 = [26.9283, 29.3067, 28]';%[kg/s] 31-33
%gas rate from reservoir
w_rg0 = [26.9283, 35.1680, 28]';%[0.1 kg/s] 34-36
%riser head pressure
p_rh0 = 22.9558;%[bar] 37
%mixture density in riser
rho_r0 = 1.3618;%[100 kg/m3] 38
%manifold pressure
p_m0 = 32.8920;%[bar] 39
%riser head total production rate
w_pr0 = 63.4446;%[kg/s] 40
%riser head total oil production rate
w_to0 = 56.2350;%[kg/s] 41
%riser head total gas production rate
w_tg0 = 7.2096;%[kg/s] 42

%%setting diff states as algebraic
%gas holdup @ annulus
m_ga0 = [1.0568, 1.0606, 1.0644]';%[ton] 43-45 mga(2) +(mga(2)-mga(1))
%gas holdup @ well
```

```
m_gt0 = [0.7470, 0.7956, 0.8442]';%[ton] 46-48 mgt(2) + (mgt(2)-mgt(1))
%oil holdup @ well
m_ot0 = [6.3000, 5.8047, 5.3094]';%[ton] 49-51
%gas holdup @ riser
m_gr0 = 0.1265;%[ton] 52
%oil holdup @ riser
m_or0 = 0.9863;%[ton] 53
%particle impact velocity
V_p0 = [2,2,2]'; % 54-56
%mixed dynamic viscosity
mu_f0 = [0.001,0.001,0.001]'; % 57-59
%g1
g10 = [0.2,0.2,0.2]'; % 60-62

%% Inputs
%gas lift rate
w_gl0 = [0.5,0.5,0.5]'; %[kg/s]

dx0 = vertcat(ER0);
z0 = vertcat(p_ai0,p_wh0,p_wi0,p_bh0,rho_ai0,rho_m0,w_iv0,w_pc0,w_pg0,...
w_po0,w_ro0,w_rg0,p_rh0,rho_r0,p_m0,w_pr0,w_to0,w_tg0,m_ga0,m_gt0,m_ot0,...
m_gr0,m_or0,V_p0,mu_f0,g10);
u0 = w_gl0;
```

Listing 5: Code for the gas lift parameters

```
function par = ParametersGasLift(n,sandArray)

%number of wells
par.n_w = 3;
%gas constant
par.R = 8.314; %[m3 Pa/(K mol)]
%molecular weigth
par.Mw = 20e-3; %[kg/mol] -- Attention: this unit is not usual

%% Properties
%density of oil - dim:  nwells x 1
par.rho_o = 8*1e2; %[kg/m3]
%riser oil density
par.rho_ro = par.rho_o; %[kg/m3]
%1cP oil viscosity
par.mu_oil = 1*0.001; %[Pa s or kg/(m s)]

%% Project
%well parameters - dim:  nwells x 1
%length
par.L_w = [1500;1500;1500]; %[m]
%height
par.H_w  = [1000;1000;1000]; %[m]
%diameter
par.D_w = [0.121;0.121;0.121]; %[m]
%well transversal area
par.A_w = pi.*(par.D_w/2).^2;%[m2]

%well below injection - [m]
par.L_bh = [500;500;500];
par.H_bh = [500;500;500];
```

```
par.D_bh = [0.121;0.121;0.121];
par.A_bh = pi.*(par.D_bh/2).^2;%[m2]

%annulus - [m]
par.L_a = par.L_w;
par.H_a = par.H_w;
par.D_a = [0.189;0.189;0.189];
%volume of the annulus
par.V_a = par.L_a.*(pi.*(par.D_a/2).^2 - pi.*(par.D_w/2).^2); %[m3]

%riser - [m]
par.L_r = 500;
par.H_r = 500;
par.D_r = 0.121;
%riser areas
par.A_r = pi.*(par.D_r/2).^2;%[m2]

%injection valve characteristics  - dim:  nwells x 1
par.C_iv = [0.1e-3;0.1e-3;0.1e-3];%[m2]
%production valve characteristics  - dim:  nwells x 1
par.C_pc = [2e-3;2e-3;2e-3];%[m2]
%riser valve characteristics
par.C_pr = [10e-3];%[m2]
%parameter to account for differences in gas and liquid pressures
par.slip_real = 1;

%parameters
%reservoir pressure
par.p_res = [150;155;160]; % [bar]
%Annulus temperature
par.T_a = [28+273;28+273;28+273]; %[K]
%well temperature
par.T_w = [32+273;32+273;32+273]; %[K]
%riser temperature
par.T_r = 30+273; %[K]
%separator pressure
par.p_s = 20; %[bar]

%% This one is mine to check

par.T = 86400;   % Sampling time in seconds


%% Reservoir parameters
%System parameters for nominal model
par.GOR = [0.10;0.12;0.11];
par.PI = [5;5;5];

%% For scaling the noise
% pressure meters = 1
% flow meters = 0.1
par.scale = [1,1,1,1,1,1,0.1,0.1,0.1,0.1,0.1,0.1,1,1,0.1,0.1]';

%% For erosion model
% Sand
par.d_p = 2.5*10^(-4); %[m] particle diameter
par.rho_p = 2.5*10^3; %[kg/m3] particle density
par.mdot_p = sandArray(n+1); %[kg/s] sand rate
```

```
% Choke
par.K = 2*10^(-9); %[-] material erosion constant
par.rho_t = 7800; %[kg/m3] sensity CS
par.r = 0.2; %[m] radius of curvature
par.D = 0.05; %[m] Gap between body and cage
par.H = 0.15; %[m] Height of gallery

% Constants
par.C_unit = 1000; % Unit conversion factor: now in mm/s
par.C_1 = 1.25; %[-] Model/geometry factor
par.n = 2.6; %[-] Velocity coefficient
par.GF = 2; %[-] Geometry factor

% Precalculations of erosion in choke:
par.alpha = atan(1/sqrt(2*par.r));
par.F = 0.6*(sin(par.alpha) + 7.2*(sin(par.alpha) - sin(par.alpha)^2))^0.6
* (1-exp(-20*par.alpha));
par.A_g = 2*par.H*par.D; %[m2] Effective gallery area
par.A_t = par.D_w(1)^2*pi/(4*sin(par.alpha)); % Area exposed to erosion
par.ER_constant = par.K*par.F*par.C_1*par.GF*par.mdot_p *par.C_unit/
(par.rho_t*par.A_t);
```

Listing 6: Code for simulation of the constant sandproduction case study

```
clear all
close all
load('hmatrix.mat');
% load('zk.mat');
% load('u_array.mat');
% load('y_meas.mat');
load('stepwise1.mat');
load('stepwise2.mat');
load('stepwise3.mat');
load('datamatrix_1.mat');

clc
addpath('/Users/SALI/Desktop/MATLAB/casadi-osx-matlabR2015a-v3.5.5')
import casadi.*

dt = 3600;
time_total = 3600*24*365; % seconds
tot_it = 500;
[x0,z0,u0] = InitialConditionGasLift_5;
x_mes_next = x0;
z_mes_next = z0;
x_next = x0;
z_next = z0;
u_k = u0;
u_opt = [];
par = ParametersGasLift;  % Constant sandproduction rate
x_vec = [];
x_vec1 = [];
obj_vec = [];
z_vec = [];
%%%%%%%%%
% ADDED %
```

```
%%%%%%%%%
y_vec = [];
tic;
nm = 70;
np = 100;
f = waitbar(0,'MPC'); % For visual display of progress
simLength = 500;
s_vec = [];
w_vec = [];
mu = 0;
sigma = 1;

%Intial erosion
e_hat = zeros(3,simLength + 2);  % +1 for the initial value and +1 for
                                                    %the final value

flag = [];
for t = 0:simLength
   waitbar(t/tot_it,f, [num2str(round(t/tot_it*100,2)),'%... Iteration '...
       , num2str(t),'...    Elapsed time: '...
   ,datestr(seconds(toc),'HH:MM:SS')]);

    %% Erosion  DIAGNOSTICS
    %%%%%%%%%%%%%%%%%
    % preprocessing %
    %%%%%%%%%%%%%%%%%
    %% Then we compute the current system measurements
    %(generated by the wellplantmodel)
    %% in the initial value generated by the initialConditionGasLift_5

    %y_next = H * z_next;   % current measurements without noise 16*1
    y_next = H * z_mes_next;   % current measurments  with noise


    y_next_append = [y_next; u_k];     %19 *1    vector
    %y_meas_append = [y_meas; u_array];         %19 * 501  matrix
    y_meas_append = [Data.yMeas{1,1}; Data.uArray{1,1}];

    %% Now we have to scale, normalization <- y_next(without erosion),
    % The next step is related to scaling the variables.
    %You need to normalize them, because the models were trained in
    %normalized data For the normalization, you need the mean and the
    %standard deviation
    % of the measurement generated by a sensor (e.g. Pbottomhole_mean,
    %Pbottomhole_std --> If you have both, you can normalize the current
    %Pbottomhole)
    % The problem is that in the beginning of the simulation, you don't
    % have enough data to properly do that (compute the mean and standard
    %deviation).
    % You need a fairly large amount of data for that.
    % Then, we propose to use the training data for normalization in the
    % begining of simulation.
    % After time reaches the threshold XXX (still needs to be determined),
    % we assume that we have enough data to properly compute the mean and
    % standard deviation and we don't need to use the training data
    % anymore.

   u_opt = [u_opt, u_k];
```

```
y_vec = [y_vec,y_next];
y_next_append_scaled = [];
mu_td = [];
std_td = [];
mu_cs = [];
std_cs = [];
xxx = 150;

if t < xxx
 mu_td = [mu_td mean(y_meas_append,2)];
 std_td = [std_td std(y_meas_append,0,2)];
 for k = 1:19
 y_next_append_scaled = [y_next_append_scaled (y_next_append(k) -...
 mu_td(k))/std_td(k)];   % scaled here in a 1*19 vector
 end

else
  y_vec_appended = [y_vec; u_opt];
  mu_cs = [mu_cs mean(y_vec_appended,2)];
  std_cs = [std_cs std(y_vec_appended,0,2)];
  for k1 = 1:19
  y_next_append_scaled = [y_next_append_scaled (y_vec_appended(k1) -...
  mu_cs(k1))/std_cs(k1)];
  end
end

%% 3. sort the scaled vector -> divide it for each well
 % for well 1,2,3 -> for 2 and 3, change index!!
 y_sc_1 = y_next_append_scaled(:,[1,4,7,10,13,14,15,16,17]);
 y_sc_2 = y_next_append_scaled(:,[2,5,8,11,13,14,15,16,18]);
 y_sc_3 = y_next_append_scaled(:,[3,6,9,12,13,14,15,16,19]);

 %% yhat is the gradient of erosion (de/dt)
 yhat1 = stepwise1.predictFcn(y_sc_1);
 yhat2 = stepwise2.predictFcn(y_sc_2);
 yhat3 = stepwise3.predictFcn(y_sc_3);

 erosionRate_hat_scaled = [yhat1; yhat2;yhat3];

 %load('errosion_array.mat');
 errosion_array = Data.erosionArray{1,1};
 % To change an array with erosions to erosion rate, we need to
 % calculate the gradient.

 errosionRate_array = gradient(errosion_array,1);

 mu_er = [mean(errosionRate_array,2)];
 std_er  = [std(errosionRate_array,0,2)];
 erosionRate_hat = [];
 for k2 = 1:3
   erosionRate_hat = [erosionRate_hat erosionRate_hat_scaled(k2)*...
       std_er(k2) + mu_er(k2)];
 end

 %% Here we transform the de/ dt data --- unto e(t)
 % using e_k+1 =e_k  + h* de/dt, where h = 1
 % We do this for well1, well2 and well3 as follows:
 e_hat(1,t+2) = e_hat(1,t+1) + 1* erosionRate_hat(1);
```

```
    e_hat(2,t+2) = e_hat(2,t+1) + 1* erosionRate_hat(2);
    e_hat(3,t+2) = e_hat(3,t+1) + 1* erosionRate_hat(3);

    % Then we extract the x_hat as follows for each well:
    x_hat = e_hat(:,t+1);

    %% Calculating input with NMPC
       [u_k,s,w_,exitflag]=  NMPC(x_hat, u_k, z_next,nm,np);
        flag(t+1) = exitflag;
    % Adding optimal input to vector
        s_vec = [s_vec,s];
        w_vec = [w_vec,w_];
      %u_opt = [u_opt, u_k];

    %% Finding the state after applying input u
     [x_next,z_next] = WellPlantModel(x_next,z_next,u_k,par);

    %% Adding noice, if active, use these in NMPC
    %x_mes_next = x_next  + normrnd(mu,sigma,[3,1]).*x0*0.015;
    z_mes_next = z_next + normrnd(mu,sigma,[62,1]).*z0*0.15;

      x_vec = [x_vec,x_next];
      x_vec1 = [x_vec1,x_hat];
      obj_vec(t+1) = sum(z_next(28:30));
      z_vec = [z_vec,z_next];
end

close(f)

%% Plotting
t = [0:1:simLength];

error1 = [];
error2 = [];
error3 = [];
for er = 1:size(x_vec,2)
  error1 = [error1 abs((x_vec(1,:) - x_vec1(1,:))/(x_vec(1,:)))*100];
  error2 = [error2 abs((x_vec(2,:) - x_vec1(2,:))/(x_vec(2,:)))*100];
  error3 = [error3 abs((x_vec(3,:) - x_vec1(3,:))/(x_vec(3,:)))*100];
end

subplot(2,1,1);
stairs(t,u_opt(1,:),'LineWidth',2);
title('Plot of the gas lift rate in kg/s');
hold on
stairs(t,u_opt(2,:),'LineWidth',2);
stairs(t,u_opt(3,:),'LineWidth',2);
legend('Well 1','Well 2','Well 3');
legend('Location','northwest');
ylim([0,2.2]);
xlabel('Time [day]');
ylabel('Gas lift rate [kg/s]');%%%

subplot(2,1,2);
plot(t,obj_vec,'LineWidth',2);
title('Plot of the total production of oil in kg/s ');
xlabel('Time [day]');
ylabel('Tot production of oil [kg/s]');
```

```
figure;


subplot(3,1,1);
plot(t,x_vec(1,:),'LineWidth',2);
title('Plot of erosion over time [days] with the real model');
hold on
plot(t,x_vec(2,:),'LineWidth',2);
plot(t,x_vec(3,:),'LineWidth',2);
legend('Well 1','Well 2','Well 3');
legend('Location','northwest');
xlabel('Time [day]');
ylabel('Erosion [mm]');

subplot(3,1,2);
plot(t,x_vec1(1,:),'LineWidth',2);
title('Plot of erosion over time [days] with the predicted model');
hold on
plot(t,x_vec1(2,:),'LineWidth',2);
plot(t,x_vec1(3,:),'LineWidth',2);
legend('Well 1','Well 2','Well 3');
legend('Location','northwest');
xlabel('Time [day]');
ylabel('Erosion [mm]');

subplot(3,1,3)
plot(t,error1,'LineWidth',2,'MarkerSize',20)
title(...
'Plot of the percentage error of the predicted model from the real model');
hold on
plot(t,error2,'LineWidth',2,'MarkerSize',20)
plot(t,error3,'LineWidth',2,'MarkerSize',20)
%axis([0 500 .5 1.5]);
xlabel('Time[day]');
ylabel('Error %');
legend('Well 1','Well 2',' Well 3');
legend('Location','northwest');
```

Listing 7: Code for simulation of the varying sandproduction case study

```


clear all
close all
load('hmatrix.mat');
%load('zk.mat');
% load('stepwise1.mat');
% load('stepwise2.mat');
load('trainedModel.mat');
% load('StepwiseExponentialData(1).mat');
% load('EnsembleExponentialData.mat');
load('datamatrix_1.mat');
%load('ExpData200TS.mat')

clc
addpath('/Users/SALI/Desktop/MATLAB/casadi-osx-matlabR2015a-v3.5.5')
import casadi.*
```

```
dt = 3600;
time_total = 3600*24*365; % seconds
tot_it = 500;
[x0,z0,u0] = InitialConditionGasLift_5;
x_mes_next = x0;
z_mes_next = z0;
x_next = x0;
z_next = z0;
u_k = u0;
u_opt = [];
x_vec = [];
x_vec1 = [];
obj_vec = [];
z_vec = [];
%%%%%%%%%
% ADDED %
%%%%%%%%%
y_vec = [];
tic;
nm = 70;
np = 100;
f = waitbar(0,'MPC'); % For visual display of progress
simLength = 500;
% For varying sand rate %0.01 was the first one
sandArray = sandproductionrate(0.02,500,'exp',0.005);
s_vec = [];
w_vec = [];
mu = 0;
sigma = 1;

%Intial erosion
e_hat = zeros(3,simLength + 2);  % +1 for initial value andfinal value

%For the varying sand production rate
%     sr_n_1 = [];
%     for i = 1:500+1
%     sr_n_1 = [sr_n_1 (sandArray(i) - mean(sandArray))/std(sandArray)];
%     end

flag = [];
for t = 0:simLength
   waitbar(t/tot_it,f, [num2str(round(t/tot_it*100,2)),'%... Iteration ',...
       num2str(t),'...    Elapsed time: '...
   ,datestr(seconds(toc),'HH:MM:SS')]);

    %% Erosion  DIAGNOSTICS

    %%%%%%%%%%%%%%%%%
    % preprocessing %
    %%%%%%%%%%%%%%%%%
    %% Then we compute the current system measurements
    %%(generated by the wellplantmodel)
    %% in the initial value generated by the initialConditionGasLift_5

    %y_next = H * z_next;   % current measurements without noise 16*1

    y_next = H * z_mes_next;   % current measurments  with noise
```

```
 % The gas lift flowrate used as one of the predictors(regressors).
 %So here, we have to add it to the "y_next"
 % (dim u_k = 3x1)
 sarray = sandArray(t+1);

 y_next_append = [sarray;y_next; u_k];
 y_meas_append = [Data.yMeas{1,1}; Data.uArray{1,1}];
 y_meas_append = [sandArray;y_meas_append];

 %% Now we have to scale, normalization <- y_next(without erosion)
 % The next step is related to scaling the variables. You need to
 %normalize because the models were trained in normalized data
 % For the normalization, you need the mean and the standard deviation
 % of the measurement generated by a sensor (e.g. Pbottomhole_mean,
 %Pbottomhole_std --> If you have both, you can normalize the current
 %Pbottomhole)
 % The problem is that in the beginning of the simulation, you don't
 % have enough data to properly do that (compute the mean and standard
 %deviation). You need a fairly large amount of data for that.
 % Then, we propose to use the training data for normalization in the
 % begining of simulation.
 % After time reaches the threshold XXX (still needs to be determined),
 % we assume that we have enough data to properly compute the mean and
 % standard deviation and we don't need to use the training data
 % anymore.

u_opt = [u_opt, u_k];
y_vec = [y_vec,y_next];
y_next_append_scaled = [];
mu_td = [];
std_td = [];
mu_cs = [];
std_cs = [];
xxx = 500;
sandr= sandArray(1,1:t+1);
%sandr= 0.01* ones(1,t+1);

if t < xxx
 mu_td = [mu_td mean(y_meas_append,2)];
 std_td = [std_td std(y_meas_append,0,2)];
 for k = 1:20
 y_next_append_scaled = [y_next_append_scaled (y_next_append(k)...
 - mu_td(k))/std_td(k)];   % scaled here in a 1*20 vector
 end
else
  y_vec_appended = [sandr;y_vec; u_opt];
  mu_cs = [mu_cs mean(y_vec_appended,2)];
  std_cs = [std_cs std(y_vec_appended,0,2)];
  for k1 = 1:20
  y_next_append_scaled = [y_next_append_scaled ...
      (y_vec_appended(k1) - mu_cs(k1))/std_cs(k1)];
  end
end


%% 3. sort the scaled vector -> divide it for each well
 % for well 1,2,3 -> for 2 and 3, change index!!
 y_sc_1 = y_next_append_scaled(:,[1,2,5,8,11,14,15,16,17,18]);
```

```
    y_sc_2 = y_next_append_scaled(:,[1,3,6,9,12,14,15,16,17,19]);
    y_sc_3 = y_next_append_scaled(:,[1,4,7,10,13,14,15,16,17,20]);

    %% yhat is the gradient of erosion (de/dt)
    %Predicting the erosion using the stepwiseexponential model
    yhat1 = trainedModel.predictFcn(y_sc_1);
    yhat2 = trainedModel.predictFcn(y_sc_2);
    yhat3 = trainedModel.predictFcn(y_sc_3);

    erosionRate_hat_scaled = [yhat1; yhat2;yhat3];

    %% Now we unormalize
     errosion_array = Data.erosionArray{1,1};

% To change an array with erosions to erosion rate, we need to
% calculate the gradient.
    errosionRate_array = gradient(errosion_array,1);
    % creats 3*1 mean of each row in errosion_array
    mu_er = [mean(errosionRate_array,2)];
    % creats 3*1 std of each row in eroosion_array
    std_er  = [std(errosionRate_array,0,2)];
    erosionRate_hat = [];
    for k2 = 1:3
      erosionRate_hat = [erosionRate_hat erosionRate_hat_scaled(k2)...
          *std_er(k2) + mu_er(k2)];
    end

    %% Here we transform the de/ dt data --- unto e(t)
    % using e_k+1 =e_k  + h* de/dt, where h = 1
    % We do this for well1, well2 and well3 as follows:
    e_hat(1,t+2) = e_hat(1,t+1) + 1* erosionRate_hat(1);
    e_hat(2,t+2) = e_hat(2,t+1) + 1* erosionRate_hat(2);
    e_hat(3,t+2) = e_hat(3,t+1) + 1* erosionRate_hat(3);

    % Then we extract the x_hat as follows for each well:
    x_hat = e_hat(:,t+1);

%     %% Calculating input with NMPC
       [u_k,s,w_,exitflag]=  NMPC(x_hat, u_k, z_next,nm,np,t,sandArray);
       flag(t+1) = exitflag;
%     % Adding optimal input to vector
        s_vec = [s_vec,s];
        w_vec = [w_vec,w_];
        %u_opt = [u_opt, u_k];

    %% Finding the state after applying input u
     par = ParametersGasLift(t,sandArray);
      [x_next,z_next] = WellPlantModel(x_next,z_next,u_k,par);
    %% Adding noice, if active, use these in NMPC
    %x_mes_next = x_next  + normrnd(mu,sigma,[3,1]).*x0*0.015;
    z_mes_next = z_next + normrnd(mu,sigma,[62,1]).*z0*0.015;

      x_vec = [x_vec,x_next];
      x_vec1 = [x_vec1,x_hat];
      obj_vec(t+1) = sum(z_next(28:30));
      z_vec = [z_vec,z_next];
end
```

```
close(f)

% Plotting
t = [0:1:simLength];

error1 = [];
error2 = [];
error3 = [];
for er = 1:size(x_vec,2)
  error1 = [error1 abs((x_vec(1,:) - x_vec1(1,:))/(x_vec(1,:)))*100];
  error2 = [error2 abs((x_vec(2,:) - x_vec1(2,:))/(x_vec(2,:)))*100];
  error3 = [error3 abs((x_vec(3,:) - x_vec1(3,:))/(x_vec(3,:)))*100];
end

subplot(2,1,1);
stairs(t,u_opt(1,:),'LineWidth',2);
title('Plot of the gas lift rate in kg/s');
hold on
stairs(t,u_opt(2,:),'LineWidth',2);
stairs(t,u_opt(3,:),'LineWidth',2);
legend('Well 1','Well 2','Well 3');
legend('Location','northwest');
ylim([0,2.3]);
xlabel('Time [day]');
ylabel('Gas lift rate [kg/s]');%%%

subplot(2,1,2);
plot(t,obj_vec,'LineWidth',2);
title('Plot of the total production of oil in kg/s ');
xlabel('Time [day]');
ylabel('Tot production of oil [kg/s]');
figure;


subplot(3,1,1);
plot(t,x_vec(1,:),'LineWidth',2);
title('Plot of erosion over time [days] with the real model');
hold on
plot(t,x_vec(2,:),'LineWidth',2);
plot(t,x_vec(3,:),'LineWidth',2);
legend('Well 1','Well 2','Well 3');
legend('Location','northwest');
xlabel('Time [day]');
ylabel('Erosion [mm]');

subplot(3,1,2);
plot(t,x_vec1(1,:),'LineWidth',2);
title('Plot of erosion over time [days] with the predicted model');
hold on
plot(t,x_vec1(2,:),'LineWidth',2);
plot(t,x_vec1(3,:),'LineWidth',2);
legend('Well 1','Well 2','Well 3');
legend('Location','northwest');
xlabel('Time [day]');
ylabel('Erosion [mm]');

subplot(3,1,3)
plot(t,error1,'LineWidth',2,'MarkerSize',20)
```

```
title(...
'Plot of the percentage error of the predicted model from the real model');
hold on
plot(t,error2,'LineWidth',2,'MarkerSize',20)
plot(t,error3,'LineWidth',2,'MarkerSize',20)
%axis([0 500 .5 1.3]);
xlabel('Time[day]');
ylabel('Error %');
legend('Well 1','Well 2',' Well 3');
legend('Location','northwest');
```