

Thomas Solli Koløen

TDD and Machine Learning of SEPTIC MPC application

Master's thesis in Cybernetics and robotics

Supervisor: Lars Struen Imsland

June 2022

Thomas Solli Koløen

TDD and Machine Learning of SEPTIC MPC application

Master's thesis in Cybernetics and robotics
Supervisor: Lars Struen Imsland
June 2022

Norwegian University of Science and Technology

Acknowledge

I would like to express my great gratitude to the inspiring supervisors from Equinor, John-Morten Godhavn, Pål Kittilsen and Einar S.Idsø. Throughout the master thesis they have helped and challenged me, and at all time been available for me.

I would like to further express my gratitude to my supervisor at NTNU, Lars Struen Imsland, for excellent support.

20.06.2022 Thomas Solli Koløen

Abstract

In autumn 2021, the specialization project [1] successfully implemented functionality with the use of TDD with Equinor’s in-house MPC application called SEPTIC. The master thesis takes the specialization project further, investigating how more complex functionality and system dynamics can be developed with TDD. The test setup from the specialization project was used, and a sequence involving system dynamics was chosen to be developed with TDD.

The sequence was successfully implemented with the use of TDD. It was found that sequences in SEPTIC are pretty tedious to code, and a suggestion for Equinor to implement SR latch function in their configuration file, would make the implementation of the sequence much less tedious. In addition, TDD of system dynamics was successfully developed. An essential contribution of this is a *step_until()* function, and accessing simulation from ring buffer in SEPTIC. These two contributed to complete control of the simulation by stopping the simulation when a certain wanted value is achieved, and further accessing simulation to test the system dynamics in pytest.

A machine learning algorithm with LSTM cells was made. It was trained using a training dataset where the choke opening is between 15% – 100%. The ML model performed well on new unseen data on the test dataset and accurately predicted steady-state oil rates. A model which is trained in all areas of operation would be preferable, but this would be at the expense of the model’s performance. A gain scheduler was further implemented in SEPTIC. The gain scheduler uses calculated gains from the Machine learning (ML) model and further interpolates datasets of gains to a gain function. Implementation of the gain scheduler showed slightly better performance reaching set-point a bit faster. A possible implementation of a running ML algorithm along with SEPTIC was discussed. Further, one of the advantages of having an ML algorithm running beside SEPTIC to predict gain would be that the ML model would capture changes in system dynamics over time and always be up to date, setting the correct gain at all times.

Abstrakt

Høsten 2021 fordypningsprosjektet [1] implementerte med hell, funksjonalitet i Equinor's MPC SEPTIC ved å ta i bruk TDD. Masteroppgaven tar dette funnet videre, og ser på hvordan kompleks funksjonalitet som involverer testing av system dynamikk kan videre bli utviklet med TDD metodologien. Testoppsettet fra fordypningsoppgaven ble valgt å gjenbruke, og en sekvens som involverer testing av system dynamikk ble valgt som basis, for å se på muligheten å utvikle system dynamikk ved hjelp av TDD.

Sekvensen ble vellykket implementert ved hjelp av TDD. Det ble vist at implementering av sekvens i SEPTIC config fil, kan være kronglete, og det ble gjort et funn eller forslag til Equinor om å legge til SR latch som en funksjon i config filen deres. Slik funksjon ville gjort koden svært mye enklere og mindre kronglete å implementere sekvenser. I tillegg ble system dynamikk vellykket utviklet ved hjelp av TDD. En sentral del av dette var *step_until()* funksjonen og muligheten for å hente ut simuleringverdier fra ring buffer. De to bidro til fullstendig kontroll over simulering ved å stoppe simulering når en viss ønsket verdi er oppnådd, og deretter hente ut simuleringverdier for videre testing av system dynamikk i pytest.

En ML algoritme med LSTM celler ble laget. Den ble trent opp på en trening datasett hvor ventilåpningen var mellom 15% – 100%. ML modellen viste god ytelse på ny usett data på test datasettet, og nøyaktig predikerte steady-state olje rate. En modell som er trent på hele arbeidsområde til ventilen ville vært foretrukket, men dette ville gått på bekostning av modellen sin predikasjons evne. En gain scheduler ble videre implementert i SEPTIC. Gain scheduleren bruker kalkulerede gains fra predikasjon fra ML modellen, og videre interpolerer datasettene av gains til en gain funksjon. Implementasjon av en gain scheduler viste litt bedre ytelse når den ble brukt i simulering sammenheng, den nådde steady-state olje rate litt tidligere. Videre vil en av fordelene av en ML algoritme som kjører på siden av SEPTIC til å predikere gains, være at ML modellen vil fange endringer i system dynamikk over tid, og alltid vær oppdatert med å sette korrekt gain til alle tider.

Contents

Acknowledge	i
Abstract	ii
Abstrakt	iii
Acronyms	vii
List Of Figures	vii
List of Listings	viii
1 Introduction	1
2 Background	3
2.1 Model Predictive Control	3
2.2 SEPTIC	5
2.2.1 XV's	7
2.2.2 SEPTIC MPC Solver	9
2.2.3 Model Representation	10
2.2.4 Calc	11
2.3 Test Driven Development	12
2.3.1 Pytest	13
2.4 FMU	14
2.5 OPC	14
2.6 antiSEPTIC	15
2.7 Oil well	18
2.8 Test setup	19
2.9 Machine learning	20
2.9.1 Feedforward Neural Networks	22
2.9.2 Recurrent Neural Network	24
2.9.3 LSTM	24
2.9.4 Training loop	25
2.9.5 TensorFlow	25
3 Sequence	26
3.1 Sequence implementation	26
3.1.1 Well Off	27
3.1.2 Well Closed	29
3.1.3 Well Kick Off	31
3.1.4 Full Sequence Implementation	33
3.2 Simulation	33

4	Machine Learning of System Dynamics	35
4.1	Learning problem	35
4.2	Dataset	35
4.2.1	Training set	36
4.2.2	Test set	37
4.2.3	Convert data	38
4.2.4	Data prepossessing	38
4.3	Create and Train a Machine Learning Model	40
4.4	Evaluation of model	42
4.5	Gain scheduling implementation in SEPTIC	45
4.6	Simulation on sequence	47
5	Discussion	48
5.1	Sequence Improvement	48
5.2	TDD of system dynamics	48
5.3	Machine learning of system dynamics	49
6	Conclusion	51
	References	52
A	Appendix A: Python code	54
A.1	step_until.py	54
A.2	confptest.py	54
A.3	test_sequence.py	55
A.4	test_testSet.py	60
A.5	Convert_Data.py	61
A.6	ReadWriteFile.py	62
B	Appendix B: SEPTIC configuration files	62
B.1	WellKickOff calc	62
B.2	Well Sequence calc	64
B.3	SinglWell.cnfg	68
C	Appendix C: Machine Learning python code	84
C.1	ML_import.py	84

Acronyms

Notation	Description
APC	Advanced Process Control.
BHP	Bottom Hole Pressure.
CSV	comma-separated values.
CV	Control Variable.
DV	Disturbance Variable.
EV	Evaluated Variable.
FFNN	Feedforward Neural Network.
FMI	Functional Mock-up Interface.
FMU	Functional Mock-up Unit.
GUI	Graphical User Interface.
HMI	Human Machine Interface.
LSTM	Long short-term memory.
MAE	Mean Absolute Error.
MIMO	Multiple-input and Multiple-output.
ML	Machine learning.
MPC	Model Predictive Control.
MSE	Mean Square Error.
MV	Manipulated Variable.
NLP	Nonlinear Program.
NMPC	Nonlinear Model Predictive Control.
OLE	Object Linking and Embedding.
OPC	Open Platform Communication.
PID	Proportional–Integral–Derivative.
PLC	Programmable Logic Controller.
QP	Quadratic Program.
ReLU	Rectified Linear Unit.
RNN	Recurrent Neural Network.
RTO	Real Time Optimization.

Notation	Description
RUI	Remote User Interface.
SEPTIC	Statoil Estimation and Prediction Tool for Identification and Control.
SISO	Single-input and Single-output.
SR	set-reset.
TDD	Test-Driven Development.
TV	Trend Variable.
UA	Unified Architecture.
WHP	Well Head Pressure.

List of Figures

1	Illustration of the MPC principle Source: Figure 4.1 from [4]	5
2	The graphical user interface of Statoil Estimation and Prediction Tool for Identification and Control (SEPTIC)	6
3	Illustration of hierarchy of objects in SEPTIC	7
4	Illustration of SEPTIC-notation	8
5	Screenshot from SEPTIC Graphical User Interface (GUI) of an Manipulated Variable (MV)	8
6	Screenshot from SEPTIC GUI of an Control Variable (CV)	9
7	Screenshot from SEPTIC user interface for experimental model representation	11
8	Implementation of a Calc in a configuration file	11
9	Test-Driven Development (TDD) methodology cycle	12
10	Typical OPC communication topology	15
11	Illustration of how antiSEPTIC can be incorporated in a test setup with pytest and SEPTIC	15
12	Illustration of an oil well process	18
13	Step responses of a choke	19
14	Test setup Illustration	20
15	How a neuron looks like mathematically	22
16	Illustration of a Feedforward Neural Network	23
17	LSTM cell structure	24
18	Illustration of the sequence	26
19	The methodology for sequence implementation	27
20	Plot of Choke and OilRate	34
21	Plot of the wellstate	34
22	Training dataset	37
23	Test dataset	38
24	Evaluation plot of training set	43
25	Evaluation plot of test set	44

26	Plot of the gain function	46
27	Simulation with and without gain scheduler	47
28	Illustration of the architecture of a ML model implementation	50

Listings

1	Code that shows how to access members in SEPTIC	16
2	Code showing how to access OPC tag's	17
3	Well Off unit test	27
4	Well Off calc	28
5	Well Closed unit test	29
6	Well Closed calc	30
7	Well Kick Off unit test	32
8	Training set python code	36
9	Data sequence conversion code	39
10	MinMaxScaler python code	39
11	Read training set, scale it and create sequences out of it	40
12	Python code for creating a ML model	40
13	Python code for training model	41
14	SEPTIC config file for gain scheduling	45
15	Code for IsWellClosed calc	48
16	Code for IsWellClosed calc with a potential set-reset (SR) latch	48
17	step_until.py	54
18	confstest.py	54
19	test_sequence.py	55
20	test_testSet.py	60
21	Convert_Data.py	61
22	ReadWriteFile.py	62
23	Well Kick Off calc	62
24	Well Sequence calc	64
25	SEPTIC configuration file	68
26	ML_import.py	84

1 Introduction

The master thesis is in cooperation with Equinor. Equinor is a Norwegian energy company operating internationally in around 30 countries worldwide. Equinor's portfolio of projects encompasses oil and gas, renewable, and low-carbon solutions [2]. The master thesis originates from the Rotvoll office in Trondheim. The Rotvoll office is in charge of the implementation and development of Equinor's in-house Model Predictive Control (MPC) called SEPTIC.

In the autumn of 2021, a specialization project investigated the possibilities of developing SEPTIC with TDD. The specialization project successfully implemented functionality to SEPTIC with TDD, but it remained unsolved wherever it can be used to develop the system dynamics, it discussed the possibility of such. The master thesis is an extension of the specialization project, and such will investigate further in practice how system dynamics can be developed.

SEPTIC is an in-house MPC, used in offshore oil production and oil refineries [3]. From 1996 to today, SEPTIC has been developed to improve performance and functionality. It is a robust and well-developed application, where functionality needed has been developed throughout the years. However, some functionality can be more cumbersome to add than others, one of which is sequences. There is possibility to add a sequence to SEPTIC through various calculation modules in SEPTIC. However, the implementation is tedious and requires some experience with SEPTIC. A method to implement sequence in SEPTIC would be beneficial for more robust, more straightforward, and faster implementation in SEPTIC. A startup and close sequence requires both developments of functionality and system dynamics. It would be good use as a basis for further exploration of the use of system dynamics development with TDD on SEPTIC, and at the same time, look into how the sequence implementation can be improved.

As implied in the name, Model Predictive Control uses a system model to predict the system's future behavior for controlling the system. A precise system model is a crucial ingredient for MPC to work optimally. The closer the model is to the actual system, the closer will the control inputs calculated at first be the right. In total, a good model gives better performance.

A model in SEPTIC is found by doing an experimental Single-input and Single-output (SISO) step response. But such model requires linearity through the whole operation area, but the linearity isn't true for the relation between choke position and oil rate. SEPTIC can deal with such non-linearity by doing gain-scheduling on the step-response model. Gain scheduling has shown to deal with non-linearity behaviour satisfactorily [3]. Calculation of the gains is typically done by doing step-responses through the whole area of operation, afterwards calculate the different gains. The thesis will look into how a ML algorithm can be used to learn the system dynamics of the system and use it in a running SEPTIC application, to automatically calculate the gains.

The system chosen to use in the thesis is a single oil well simulated as a dymola model in SEPTIC. The test environment is inherited from the specialization project but slightly modified from a three oil well application, to a single oil well application. Sections 2.7 and 2.8 describes the system and test environment completely. The system is limited to a simple SISO system, even though the dymola model has support for an additional control input (gas lift rate). Gas lift-rate will be held to a constant value through all simulations, therefore only affecting the oil rate will be the opening of the choke.

The intention of the thesis is to look into how system dynamics can be developed with TDD by implementing a running sequence. When a running sequence is up and running, the thesis will look into how a ML algorithm can be used to learn the system dynamics of a simulation. Afterwards use the ML-model to calculate gains for the gain scheduler, and implement the gain scheduler, and run a complete simulation with the gain scheduler on the sequence implemented earlier, and compare with a non-gain-scheduled simulation.

Section 2 describes some relevant background and theory used in rest of the thesis, while section 3 uses this theory to create a sequence which is developed by the use of TDD, while section 4 creates a ML algorithm and gain scheduler. In section 5 the results from sections 3 and 4 is discussed, and at last section 6 makes up a conclusion.

2 Background

This chapter presents relevant background and theory to test out TDD methodology and assemble a test environment for SEPTIC.

2.1 Model Predictive Control

MPC is an advanced control approach that falls within the Advanced Process Control (APC) branch. The benefit of MPC is that it optimizes the present time slot while also taking into account how the system develops in the future. MPC works by making a mathematical model of the process, and then predict the system, to look how the process unfolds itself. The prediction is done through solving a finite open-loop control problem. The initial control input from the solution of the control problem is then used as control inputs to the actuators at each sampling instant [4]. All together MPC can open a valve, and achieve a optimal state before the controller gets feedback from the process [5].

MPC-controller is one of the most advanced controller techniques, it can control a Multiple-input and Multiple-output (MIMO) system. Whereas a traditionally Proportional–Integral–Derivative (PID) controller is limited to a SISO system. In addition to control a MIMO system, the MPC can tune actuators and responses related to each-other, to achieve desired performance. For instance a system with two actuators and two states, one state is more valued to reach it desired set-point rather than the other. MPC can tune such the favoured state will have better response than the other. Actuators can also be tuned, such it can use less or more “energy” in the control inputs. A MPC requires to predict how the system behaves in the futures, while PID computes it’s control input based upon the error between actual value and the set-point. This yields a higher computation effort for a MPC compared to a PID. MPC can handle constraints such as limiting the responses, actuators and rate of change in actuators, constraint cannot be found in a PID controller.

The optimizing prediction problem is formulated in a finite horizon open loop control. Such problem consists of a objective function, equality constraints and inequality constraints. Solution of such is done with a Quadratic Program (QP) solver over a time horizon from $t = 0$ to $t = N$. This problem is linear, but can be adapted to a nonlinear system with a Nonlinear Program (NLP) solver. Equations (1) and (2) formulates this problem mathematically.

$$f(z) = \min_u \sum_{t=0}^{N-1} x_{dev}^T Q x_{dev} + u_{dev}^T R u_{dev} + \Delta u^T R_{\Delta} \Delta u \quad (1)$$

where

$$x_{dev} = x_{t+1} - x_{t+1}^{ref}$$

$$u_{dev} = u_{t+1} - u_{t+1}^{ref}$$

$$\Delta u = u_t - u_{t-1}$$

$$Q \succ 0$$

$$R \succ 0$$

$$\Delta R \succ 0$$

subject to

$$x_{t+1} = A_t x_t + B_t u_t \quad (2a)$$

$$x^{low} \leq x_t \leq x^{high} \quad (2b)$$

$$u^{min} < u_t < u^{max} \quad (2c)$$

$$\Delta u_{min} < \Delta u < \Delta u_{max} \quad (2d)$$

Equation (2a) is an equality constraint which constraints the optimizing problem to a model of the system to predict on. Limiting the process state, control input and rate of change in control input, is done through eqs. (2b) to (2d). These constraints are used to set boundaries to decrease wear of actuators, decrease the possibility of the system to operate at the limit and to reflect limitation in the real system.

The objective function eq. (1) consist of three terms, first term are the states taken into consideration. Here the Q matrix penalize the deviation between reference trajectory, and also penalizes between the states. A easy way to tune the Q -matrix is by make it diagonal, and set lower value for those states which is favourable, and higher values for less favourable states. Then the optimizing algorithm will find it to cost “less” to minimize the favourable state. The second term tunes how the control input affects the system. Tuning of the control input is done in relation to each-other and how aggressively control inputs used. Small values of R gives more aggressively control input, and contrary higher values of R , the controller is more reluctant to aggressively control input. Same as with the Q matrix, the R matrix is chosen diagonal. To favor one control input over the other, a higher value in the R matrix for the favoured control input must be chosen. The last term penalizes rate of change in control input. As with the other matrices, the R_Δ is chosen diagonal, and controls the rate of change in the control input. As earlier it can be chosen in relation to each-other and lower value of R_Δ in diagonal allows for higher rate of change in the system, and contrary.

In addition to the tuning matrices Q , R and R_Δ , there is a possibility to use “control input blocking” in the formulation of the open loop problem. In short term, control blocking is to use same control input over several time steps with different length of blocking length. Example a blocking with the number [1 2 4 8] will have a time horizon of 15, but only four control inputs, as it will first have a control input for one time step, then one control input for two time steps and so on. Using control input blocking has

shown to reduce run-time and computational effort [4].

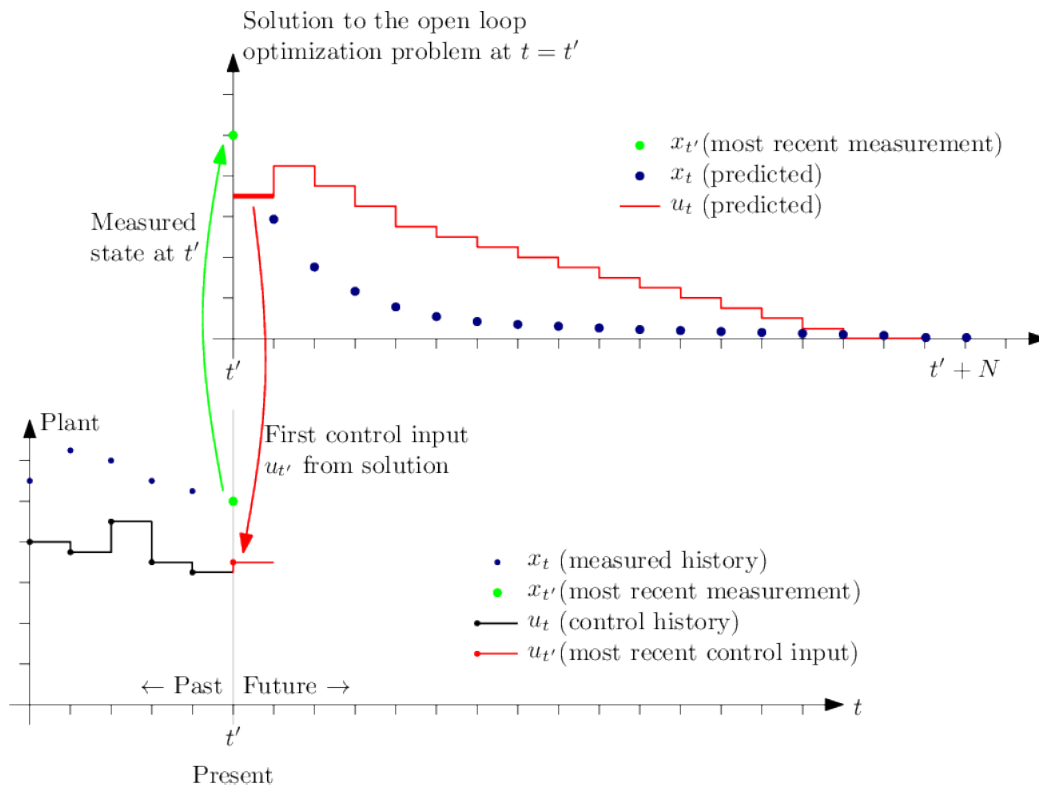


Figure 1: Illustration of the MPC principle
Source: Figure 4.1 from [4]

2.2 SEPTIC

Equinor's in-house software for MPC, Real Time Optimization (RTO), dynamic process simulation for simpler case studies, and off- and on-line parameter estimation in first principle based process models is known as SEPTIC. It has been developed since 1996, and is widely used in Equinor's process plant such as refineries and offshore production and processes. SEPTIC has been shown to provide more precise production, lower emissions, and reduced operator loads [3].

SEPTIC is an object oriented software developed in C++. SEPTIC has a GUI and a Remote User Interface (RUI) interaction. Some feature of the GUI are support for plotting of variables, set-point changes, activation of the MPC, generating and modeling of experimental models and online tuning.

A SEPTIC application is made through a SEPTIC configuration file which is read at initialisation [3]. The configuration file is object oriented, and consist of different types

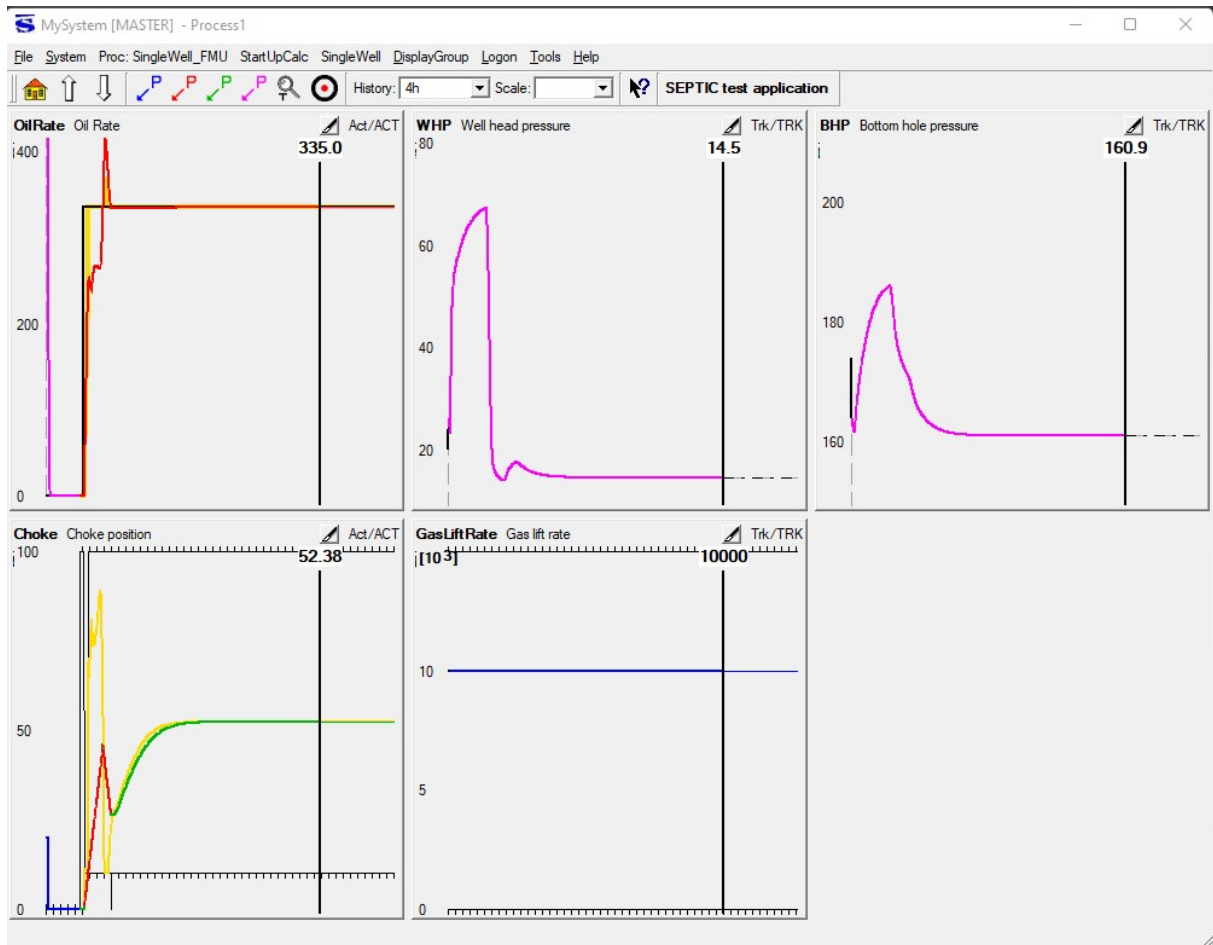


Figure 2: The graphical user interface of SEPTIC

of object. The most common and relevant object in SEPTIC are listed as followed:

- **System:** Must always be present in the configuration file, and appears only once. It controls all the calculations in the SEPTIC application. Has global parameters available for all application objects.
- **Proc:** Runs a process based upon which object implemented. Most common “Proc” object are *FMUProc.*, *SopcProc* and *MasterTcip* Which can be translated to Functional Mock-up Unit (FMU) support for process simulation, communication with a Open Platform Communication (OPC) server and RUI interface.
- **Appl:** There exist four type of application objects, the two most used “Appl”-object are *SmpcAppl.* and *DmmyAppl.* *SmpcAppl* is the application which consists of the MPC with the experimental model. *DmmyAppl* are used for data acquisition and computation. Both “Appl” objects has XV-object attached and used for either computation in MPC or data processing.
- **XV:** Used as a children in the objects described above, will be thoroughly described in section 2.2.1.

To illustrate the layout and the hierarchy of how the objects are used in a configuration file, fig. 3 illustrates a typically configuration file.

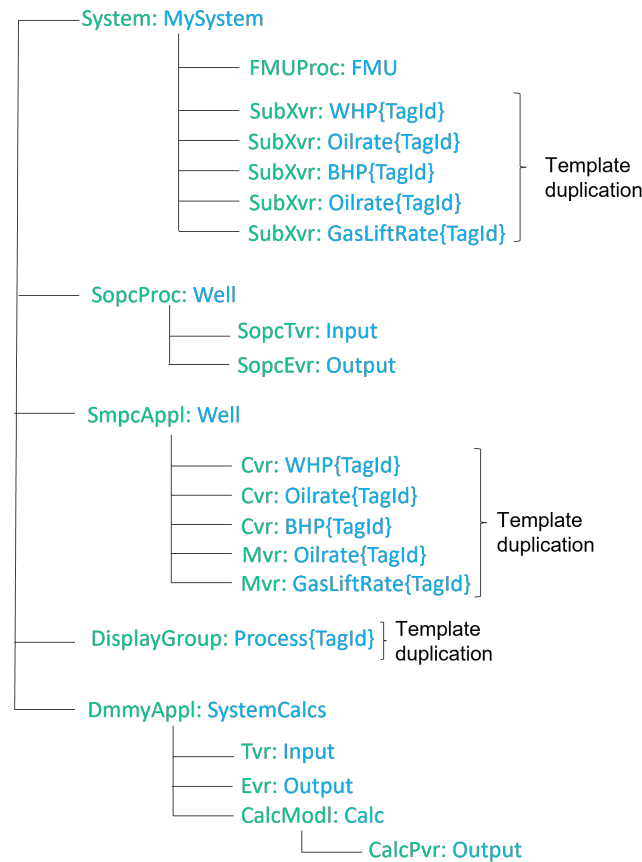


Figure 3: Illustration of hierarchy of objects in SEPTIC

2.2.1 XV's

XV is a collective term for the different variable objects used for data acquisition and data processing in SEPTIC. XV is a children object used in a application object. There exist five types of XV object that consist of different data members. They are as followed:

- **Manipulated Variable (MV):** Defined as the control input.
- **Control Variable (CV):** The measured state or response of the system.
- **Disturbance Variable (DV):** The past measured disturbance and/or estimated disturbance.
- **Evaluated Variable (EV):** Value calculated/written inside SEPTIC, can be send out from SEPTIC through OPC.

- **Trend Variable (TV):** Values that can be read from an external source for instance from a OPC-tag or a read only variable inside SEPTIC.

How the notation is used in a process, are illustrated in fig. 4.

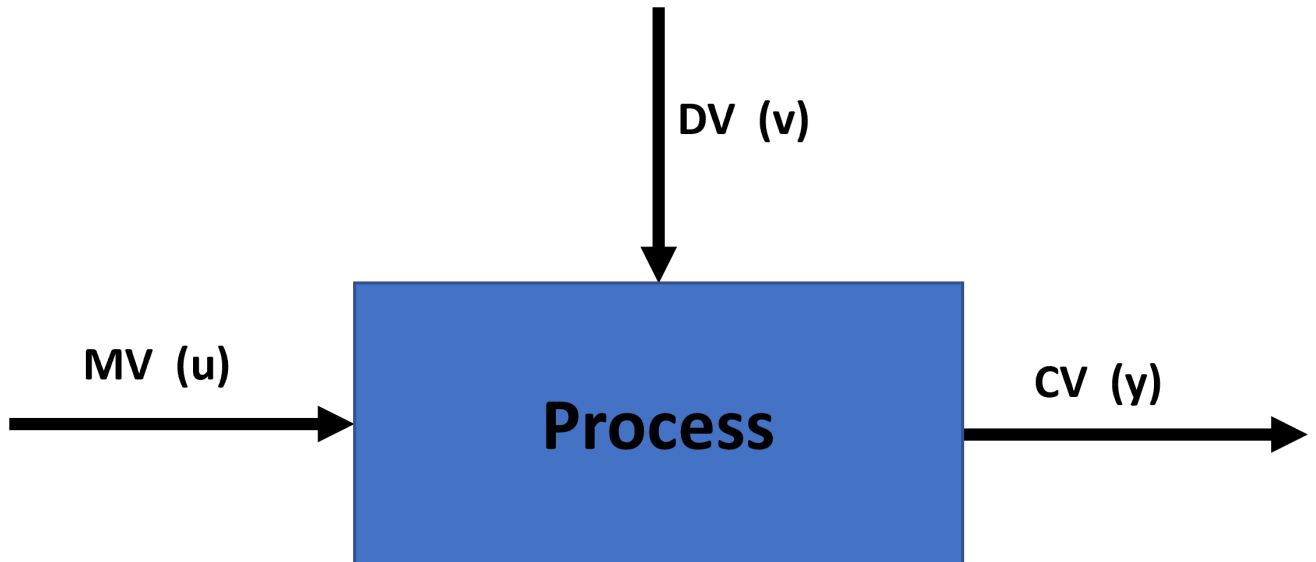


Figure 4: Illustration of SEPTIC-notation

The tuning parameters and data members in SEPTIC are dependent of which XV used. A screenshot of the MV GUI tuning parameters are shown in fig. 5. The MV has a *High* and *Low* constraints and a ideal value (*Iv*). *MaxUp* and *MaxDown* sets constraints on the max change of rate per sample in movement of the actuator, *MovePenalty* penalizes movement of actuator. *IvPrio* sets a priority of the ideal value in the steady-state solver, and *IvROC* sets the ramping of move per sample towards ideal value. *Fulf* penalizes deviation from ideal value. *Blocking* is a hidden tuning parameter configurable in *config* file.



Figure 5: Screenshot from SEPTIC GUI of an MV

In compliant with the MV, a CV has *High* and *Low* limit, but has a set-point (*SetPnt*) the MPC control against. *HighPrio*, *LowPrio* and *SetPntPrio* sets priority of the control targets in steady-state solver. Whereas *HighPenalty* and *LowPenalty* penalizes breaking of the high and low limits. *Fulf* penalizes deviation from set-point. There are two 1.order low-pass filter *BiasTfilt* and *BiasTpred*. *BiasTfilt* sets the time constant of the filter for model against process measurement, while *BiasTpred* sets the time constant of the filter for bias predication.

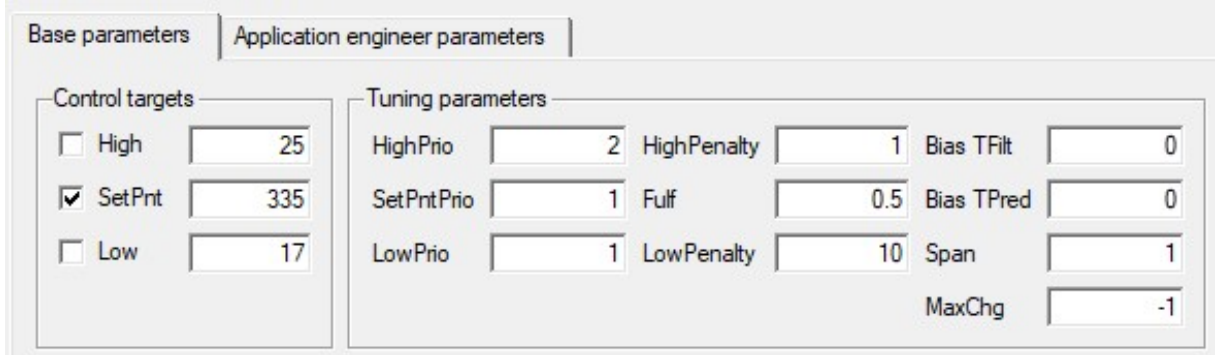


Figure 6: Screenshot from SEPTIC GUI of an CV

In addition to the object specific data members specified, there are some common data member, applicable for all XV's, most mentioned worthy are:

- *Meas*: Measured plant value
- *Mode*: Sets the desired state of the variable. Mode set to *ACTIVE*, the MPC application will take the variable object into consideration in the computation, while set to *Off* the XV is not used in the MPC application computation.
- *Span*: Scaling of variables. For instance used in tuning variables such as *Fulf* and *MovePenalty*.
- *MaxChg*: Maximum allowance of change per time, in central control target such as *High*, *Low* and *SetPnt*. Inhibits rapid changes in control targets for smoother transition.
- *GrpMask*: Defines different membership of XV's in sub applications. Allows grouping for XV, such that multiple system processes can be controlled with the same MPC application.

2.2.2 SEPTIC MPC Solver

SEPTIC sets up following finite horizon control problem:

$$\min_u \sum_{t=0}^{N-1} (y - y_{ref})^T Q_y (y - y_{ref}) + (u - u_{iv})^T Q_u (u - u_{iv}) + \Delta u^T P \Delta u + \alpha^T A \alpha \quad (3)$$

subject to

$$y = M(y, u, d, v) \quad (4a)$$

$$y_{low} - \alpha_l \leq y \leq y_{high} + \alpha_h \quad (4b)$$

$$u_{min} < u_t < u_{max} \quad (4c)$$

$$\Delta u_{min} < \Delta u < \Delta u_{max} \quad (4d)$$

Experimental SISO step response models are used to find the system model ($y = M(y, u, d, v)$). The CV's prediction horizon is computed automatically from the step-response models, giving the CV sufficient time to achieve steady-state [3]. The constraints and tuning parameters found in XV's can be translated and found in eqs. (3) and (4). The MPC solvers control priorities are as followed:

1. MV rate of change.
2. MV high/low limits.
3. CV hard constraints.
4. CV soft constraints, CV set-points, MV ideal values: Priority level 1
5. CV soft constraints, CV set-points, MV ideal values: Priority level 2
6. CV soft constraints, CV set-points, MV ideal values: Priority level n

2.2.3 Model Representation

A model of the system is done through SISO step response models. They are easy to build, understand and maintain [3]. They are linearly and based upon superposition principle, but represent the process dynamics sufficient for the use [3]. SEPTIC has dedicated functionality for auto-generating of experimental models. The model editor can generate, change and activate models while online on the controller. The models generated are saved as a file, and read at initialisation, such that the models doesn't vanish [3].

For a system which is non-linear, SEPTIC can use a non-linear model and Nonlinear Model Predictive Control (NMPC). However, gain-scheduling is an approach that seems to compensate for the non-linearity sufficient [3]. For generating a gain-scheduler, one use a calc to generate and change the experimental model's gain, based upon a graph of calculated gains. The graph is calculated based upon experimental results of step responses from the whole working area of the actuators. As the actuator works at different working areas, the calc will calculate different gains from the position of the actuator, and then set different gains on the experimental model.

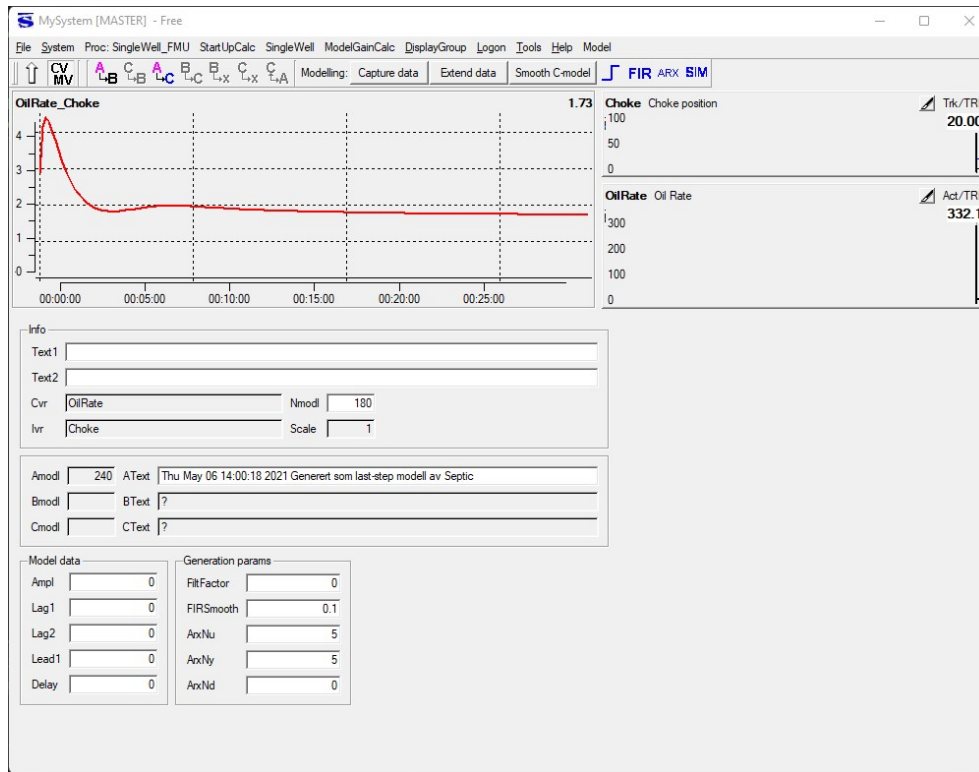


Figure 7: Screenshot from SEPTIC user interface for experimental model representation

2.2.4 Calc

A “Calc” allows to freely compute, calculate and program SEPTIC as wanted. A calc is used in a DmmyAppl, it consists of XV’s to calculate from and a *CalcModl*. The *CalcModl* consist again of *CalcPvr* object, which is where the calculation is done. To write to a EV, a *CalcPvr* must be defined inside the DmmyAppl, the *CalcPvr* and EV must be defined with the same variable name. In SEPTIC there is only EV’s that can be written to from a *CalcPvr*. But the *CalcPvr* can use all XV and other data members available in it’s algorithm calculation.

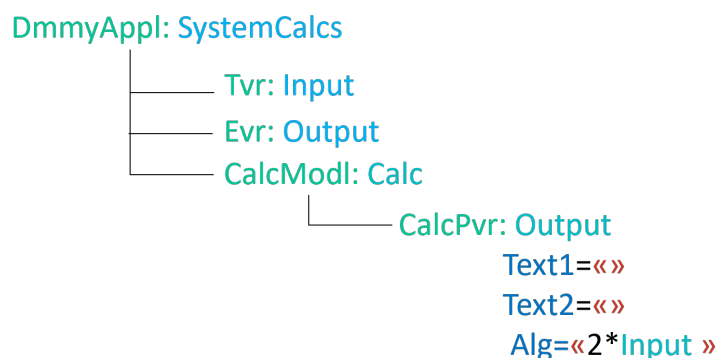


Figure 8: Implementation of a Calc in a configuration file

2.3 Test Driven Development

TDD is a software development methodology in which a test case is written before the code is implemented. The TDD methodology is straightforward and can be summarized in three steps:

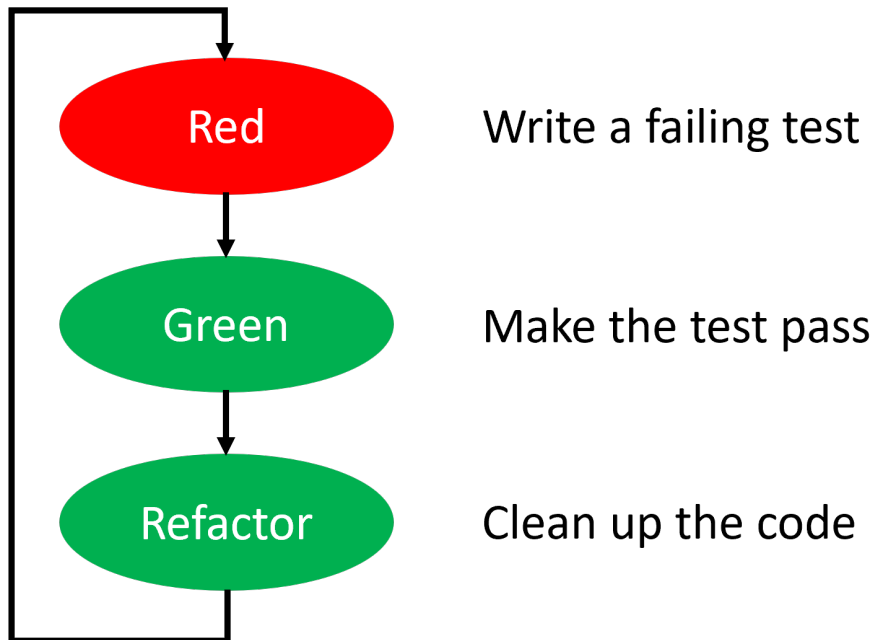


Figure 9: TDD methodology cycle

1. Red: Write the function's simplest unit test case and make sure it fails.
2. Green: To pass the unit test, implement the function in the code.
3. Refactor: Clean up and simplify the code as much as feasible..

Why should developers utilize the TDD technique? Wouldn't developing unit test cases be more work? It appears to generate even more work effort, but TDD guarantees that the code is kept to a bare minimum, which implies a simpler, better and smarter code. In addition, to the fact that software is always thoroughly tested after being implemented. Therefore testing is much less work afterward.

Typically, a developer would write spaghetti code to implement the specs, and then test the code with an integration test. It either succeeds or fails the integration test, but if it fails, there's no way of knowing where in the code it went wrong [6]. It might be an error with a function or integration. TDD eliminates this uncertainty by testing and checking the function before implementing it, and then testing the implementation again. Overall, this makes it simple to locate the error in the code [6].

Different testing methodologies are given below, with TDD being under the developer tests branch:

- **Developer tests:** Unit tests are run automatically before or after functionality is implemented.
- **Unit testing:** An automatic and/or manual test of a specific application unit.
- **Integration testing:** Tests involving two or more units operating together are performed manually and/or automatically..

Although TDD, unit testing, and developer testing appear to be fairly similar, there are several key distinctions. In simple words, unit testing and developer testing are tests of a single piece of code that is separated from the rest of the system. Isn't this the same thing as TDD? Yes, it is correct. The aim, though, is what sets them apart. The purpose of unit testing and developer testing is to see if the code works as expected. TDD, on the other hand, is primarily intended to be used as a technique to build software rather than to test it.

Writing unit tests before implementing functionality alters the developer's programming style. As a result, the code is more testable, readable, and intuitive. This is because the programmer is urged to isolate the functionality of the code into classes and functions in order to test it by using the fewest unit tests feasible.

2.3.1 Pytest

Pytest is a robust testing framework for developing small unit tests in Python while also supporting extensive functional testing for applications and libraries [7]. Test fixtures in the pytest package allow you to configure the test environment before running the test. The fixture enables the tests to run consistently and generate reproducible findings [8]. Fixtures are provided by Pytest and may be found in the `conftest.py` file. The test can always call fixtures in `conftest.py` as long as the unit tests are in the same folder. Some features of fixtures, according to the pytest website are [8]:

- Fixtures have explicit names and are activated by declaring their use from test functions, modules, classes or whole projects.
- Fixtures are implemented in a modular manner, as each fixture name triggers a fixture function which can use other fixtures.
- Fixture management scales from simple unit to complex functional testing, allowing to parameterize fixtures and tests according to configuration and component options, or to re-use fixtures across function, class, module or whole test session scopes
- Teardown logic can be easily, and safely managed, no matter how many fixtures are used, without the need to carefully handle errors by hand or micromanage the order that cleanup steps are added.

A unit test is generally composed of one or more *asserts* statements. The statement determines if the observed behavior matches the predicted behavior. The test will fail if such behaviors do not match. Making a unit test with too few assert statements can lead

to the test not covering all it should. However, don't use too many asserts statements, as this can complicate the unit test.

Even though implementing a unit test should reduce implementation problems, there is no assurance that you will write faultless code. There may be some logic missing from the unit test. The unit test may pass, but the code itself contains flaws, giving the impression that perhaps the code is of poor quality. Such consideration must the developer take into consideration, and have a reasonable coverage of what to include in unit tests.

2.4 FMU

FMU is a program that implements the Functional Mock-up Interface (FMI). FMI establishes a set of standards for model interchange across simulation and modeling software. The standard specifies a C interface that can be implemented by an executable FMU [9]. Models from Open Modelica may be used by other modeling and simulation systems thanks to the interface. In our scenario, the processes are modelled in Dymola and assembled into an FMU, which SEPTIC utilizes to simulate the process.

2.5 OPC

OPC is a widely used industrial automation communication protocol standard. It was designed to act as a "middle man" for Windows-based software and process control devices such as Programmable Logic Controller (PLC). The standard was originally called as Object Linking and Embedding (OLE) for Process Control (OPC) and was only for Windows operating systems when it was first released in 1996. It is now known as OPC classic. OPC classic has its own specification definitions for process data (OPC DA), alarms (OPC AE), and historical data (OPC HDA) [10].

In 2008, the OPC Foundation deemed the OPC classic protocol obsolete and developed the OPC Unified Architecture (UA). OPC UA is a platform-agnostic service-oriented architecture that combines all of the capability of OPC Classic specifications into a single framework while also allowing for much more [11]. As illustrated in fig. 10, a typical OPC configuration comprises of a PLC and an OPC client coupled to an OPC server. Both the PLC and the OPC client will be able to read and write values to the OPC server, allowing software applications to communicate with industrial controllers.

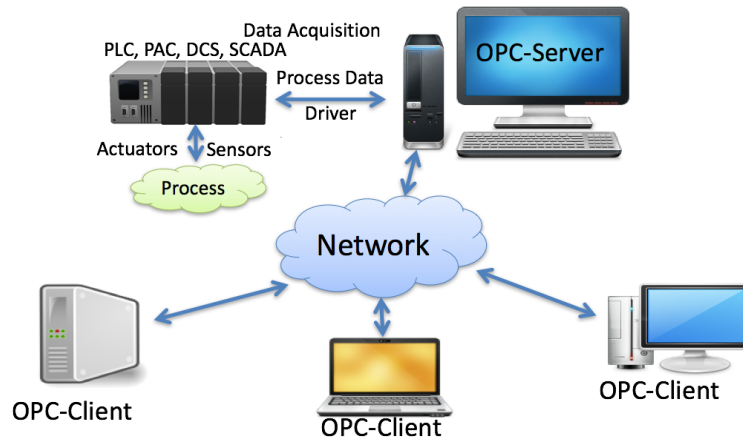


Figure 10: Typical OPC communication topology

2.6 antiSEPTIC

Equinor’s software antiSEPTIC, is a testing framework for SEPTIC. It is a recently built framework that consists of a set of classes and functions that can be easily integrated with pytest. It can be used for testing of functionality, TDD and simulation. The test framework gives complete control of simulation and accessing data in both OPC server and SEPTIC. Figure 11 illustrates how the antiSEPTIC framework is typically used in a complete test environment setting.

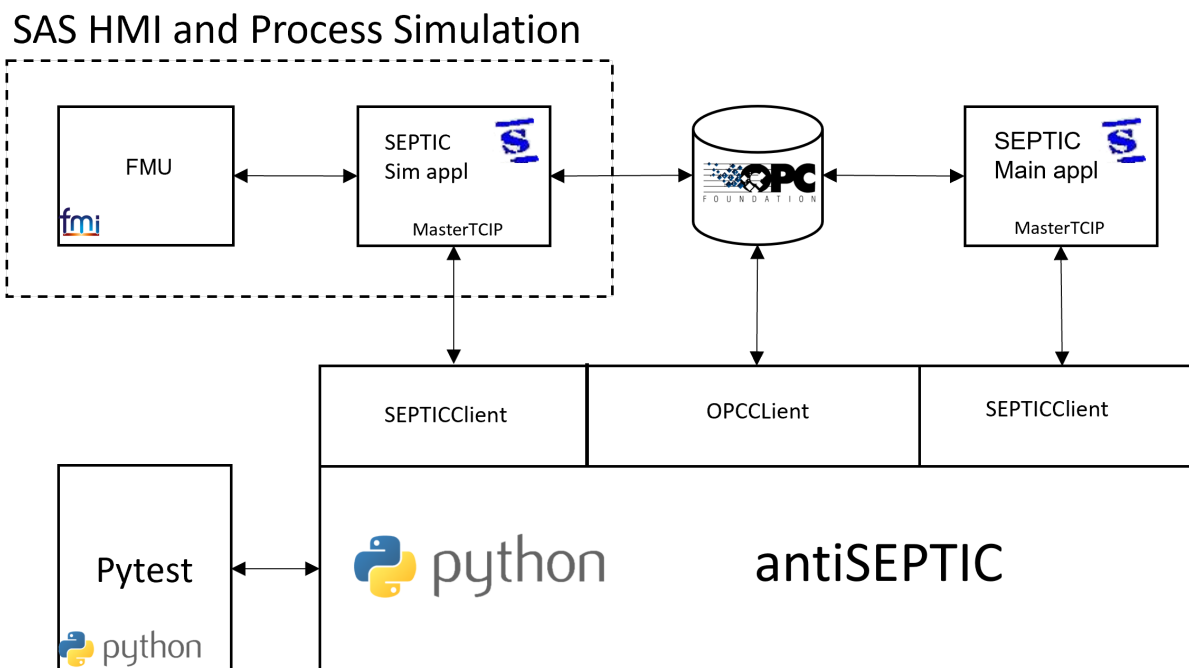


Figure 11: Illustration of how antiSEPTIC can be incorporated in a test setup with pytest and SEPTIC

There are five types of objects of importance, they are as followed:

SEPTICMaster: Initialize, start, and stop a SEPTIC master instance with this class. The parameters “cnfgfile” point to the SEPTIC config file, “qtseptic” to the SEPTIC executable file, “rundir” to the SEPTIC run directory, and “affinity” to the SEPTIC instance’s CPU affinity. The functions of the class are:

- *start()*: If SEPTIC isn’t already executing, start a new SEPTIC master instance with "cnfgfile" as configuration file.
- *stop()*: Terminate the master SEPTIC instance.

SEPTICClient: Creates a SEPTIC client that uses the RUI protocol to connect with a SEPTIC master. The class allows you to read/write data that the OPC-server normally does not allow. The functions of the class are:

- *connect(host, port)*: With the specified host and port specifications, connects to a SEPTIC MasterTcip.
- *flush()*: Reads all messages but will not treat them.
- *read()*: Read a single message if available.
- *readall()*: Reads and processes all unprocessed messages.

As mentioned in section 2.2, SEPTIC is built in an object oriented matter, with a hierarchy. antiSEPTIC adopts this hierarchy and creates objects similar to the one in SEPTIC in python. SEPTIC sends an initial message with the build of how the SEPTIC objects are built up. The SEPTICClient object reads this initial message with the *read()* function, and auto generates python objects of the SEPTIC hierarchy. Reading and writing to SEPTIC object members is done through accessing members in the SEPTICClient object, as shown in listing 1

Listing 1: Code that shows how to access members in SEPTIC

```
ReadValue=rui.appl["SystemCalcs"]["Output"].Meas # Read value
rui.appl["Well"]["Oilrate"].Mode=3 # Write value
```

OPCServer: Initialize, run, and stop an OPC server with this class. The parameters “cfgfile” point to the OPC server’s “taglist” file, “opcserver” to the OPC server executable file, and “affinity” to the OPC server instance’s CPU affinity. The functions of the class are:

- *start()*: If an OPC server instance with the same “taglist” is not already running, start a new one.
- *stop()*: Terminate the OPC server.

OPCClient: A class for interacting with an OPC server. It generates a client to connect to the OPC server and includes several support functions for controlling it. The arguments “scheduletag” (tagname for SopcProc.ScheduleTag, default to "time"), “pulstag” (tagname for SopcProc.Pulstag, default to "Heartbeat"), and “steplen” (how many seconds to increase “scheduletag” when calling the step()-function) are used to initialize the object. The functions of the class are:

- *connect(opc_server, opc_host, timeout)*: Attempts to connect to the OPC server using the specified parameters until the timeout parameter is exceeded.
- *step(numsteps, timeout)*: Increases the “ScheduleTag” on the OPC server to perform SEPTIC stepping.

The OPCClient class also has the ability to read and write to OPC server tags. The OPCClient object includes members of all tags in the OPC server, and reading/writing them is as simple as accessing the members of the OPCClient object, as shown in listing 2.

Listing 2: Code showing how to access OPC tag’s

```
Input=opc["Input"] # Read value of the "Input" tag on the OPC server
opc["Input"]=Input+1 # Increase "Input" tag with value of 1 on the OPC server
```

Directors: There are two classes in this package: *DirectorSingle* and *DirectorMainSim*. Both classes handle OPCClient and SEPTICClient cooperation, making it easier to complete a step. To put it another way, while using *opc.step()*, one must also use *RUI.read()* to read an auto-generated calculation message sent from SEPTIC to RUI. As a result, the director class will do an *opc.read()* and an *opc.step()* for each step called in *director.step()*. The distinction between *DirectorSingle* and *DirectorMainSim* is that *DirectorSingle* only take care of one SEPTIC instance, whereas *DirectorMainSim* take cares of two. The functions of both classes are the same, and are as followed:

- *step(numsteps, timeout)*: Attempts to complete the number of steps specified in “numsteps”.
- *bootstrap()*: Steps for a long enough period such that SEPTIC can write to the OPC server.

2.7 Oil well

An oil well was chosen as the process to be regulated by SEPTIC. Figure 12 depicts the process inputs and outputs for an oil well. There are two actuators regulating the flow and pressure margin across the system. The production choke and gas lift choke are the actuators, with the production choke being set by an opening between 0% and 100% and the gas-lift choke being set by a gas lift rate $[\frac{Sm^3}{hr}]$. Oil rate $[\frac{Sm^3}{hr}]$, Well Head Pressure (WHP) $[bar]$, and Bottom Hole Pressure (BHP) $[bar]$ are the measured outputs.

Gas-lifting is an artificial-lift technique in which gas is introduced to lower BHP. The decrease in BHP pressure might facilitate production or allow for higher production rates. The main valve controlling the opening (flow) flowing through the well is the production choke. In accordance to Bernoulli's principle [12], a faster moving fluid will give a decrease in pressure. And vice versa an increase in pressure yield an increase of fluid velocity. In order for the well to produce, there must be sufficient pressure margin over the production choke, such that the fluid moves. A fully opening of the production choke will give a decrease in pressure, and increase in flow through the well. But if there is no pressure there will not be any fluid flowing through the well. Such the gas lift rate choke, enables to pressurise the well such that fluid flowing through the well can travel with a faster velocity and with enough pressure margin. The two input variables allow you to establish a desired oil rate with the opening, while the gas-lift guarantees a necessary pressure margin over the production choke while also allowing you to control the flow.

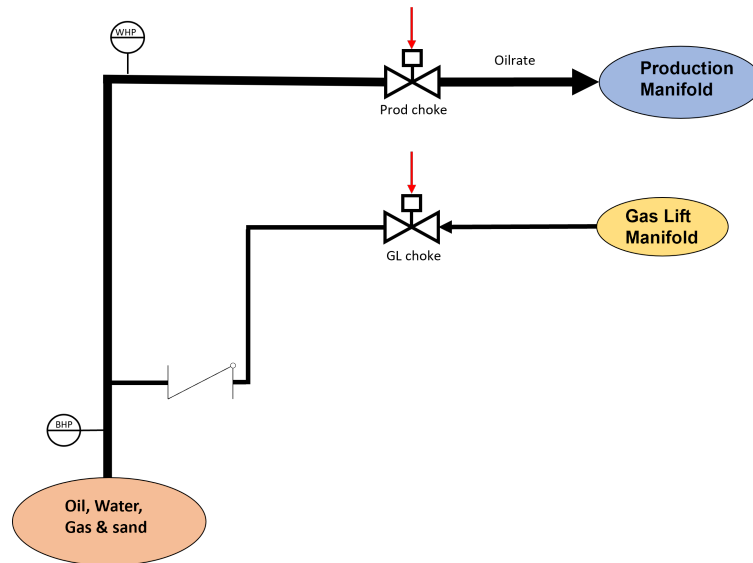


Figure 12: Illustration of an oil well process

The relation between choke opening and oil rate is nonlinear. A step response in the lower region of the choke produces a large reaction in oil rate, but a step response in the higher working area of the choke produces a significantly lesser response in oil rate.

As illustrated in fig. 13, the delta step size are the same (5%), but steady-state gain are higher for the choke opening 15 – 20 compared to 30 – 35.

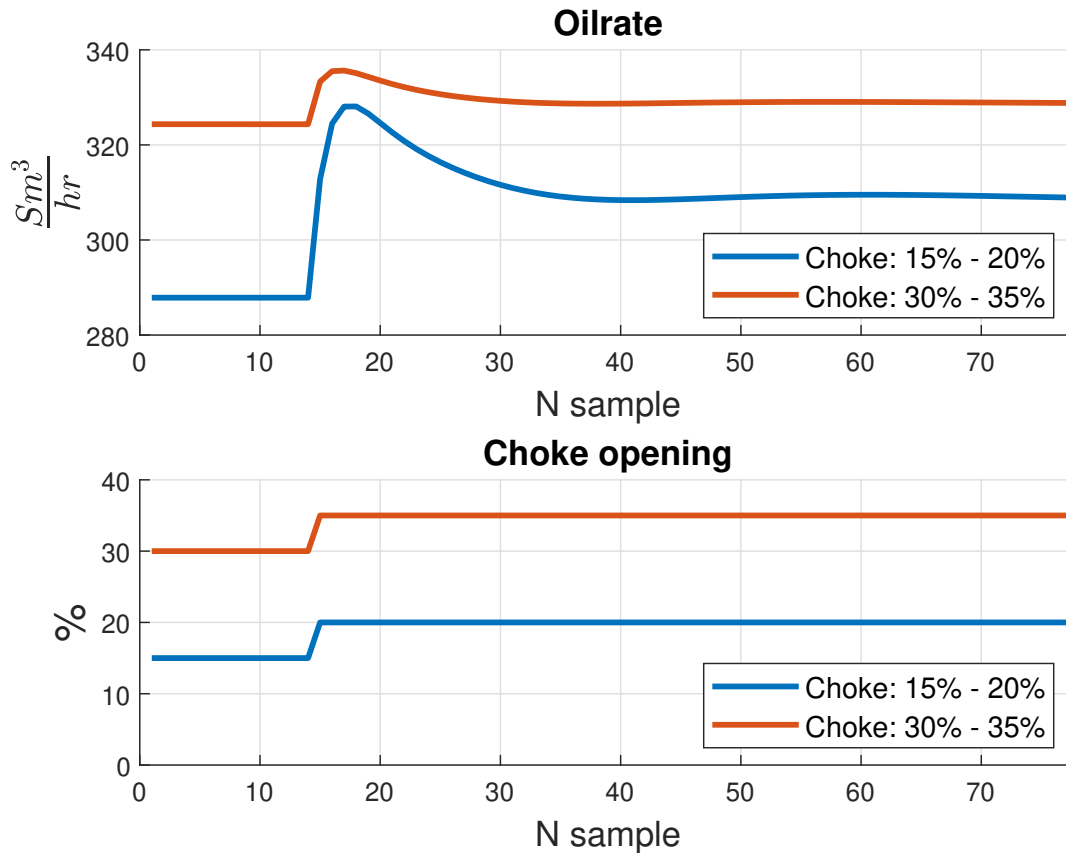


Figure 13: Step responses of a choke

2.8 Test setup

The test environment is inherited from the specialization project [1], but slightly modified. The modification includes a configuration file which includes one oil well instead of a three well. The modification are simple, the procedure are the same as section 3.1.1 in [1], but template duplication are only done once instead of three. When it comes to the pytest setup, it includes the same files as section 3.2.1 in [1], but the three well FMU file, is exchanged with a single well FMU file. A illustration of the test setup is shown in fig. 14. The configuration file is found in appendix B.3, and conftest file setting up the fixtures is found in appendix A.2.

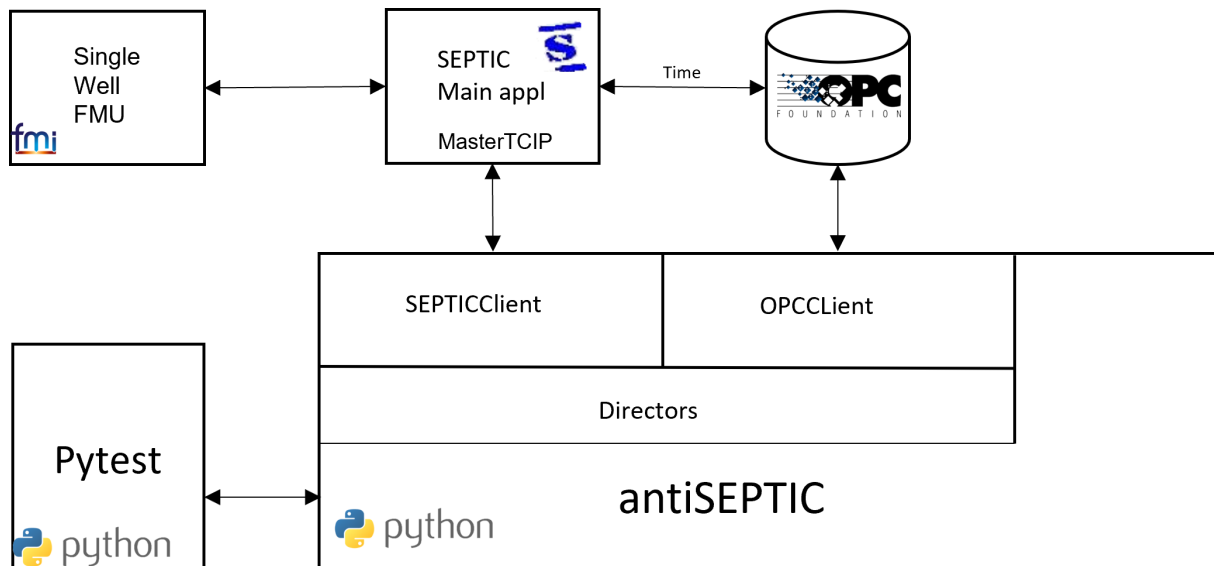


Figure 14: Test setup Illustration

2.9 Machine learning

To understand what a Machine Learning (ML) algorithm is, one must understand the purpose of it. The main purpose of a ML-algorithm is to learn from data. But what does that says in reality, well Mitchell came with a definition “A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E .” [13]. To get a grasp of what Mitchell meant, one can look at a chess of game and set up following learning problem:

- **Task T :** Playing chess.
- **Performance measure P :** How many of games won.
- **Training experience E :** Playing chess matches against itself, with experience on how chess movement affects the game.

Here a ML-algorithm that learns to play chess, might improve to play chess with performance measure of how often it wins games, by obtaining experience through playing matches against itself. Such it will improve playing chess for each time playing a game, and learn from each experience.

A task is not to learn something, but the learning itself is obtaining the ability to perform the task. For example a autonomous boat. The task is to drive the boat, you can solve it by manually program the boat to drive to the other side of the dock. But a ML-algorithm’s task is to do the same task in terms of processing an example, and improve the task based upon the example with a performance measure. The most common tasks solved by ML are:

- **Regression:** Predict a numerical value y given a data point x . It creates a functions which maps $f : \mathbb{R}^m \rightarrow \mathbb{R}$ such that $y = f(x)$
- **Classification:** Predicts which type of category the input k belongs to. It produces a function $f : \mathbb{R}^m \rightarrow \{1, \dots, k\}$, which assigns x to a class $y = f(x)$
- **Structured output:** Any task where output is a vector, with important relationship between the different elements. Example is segmentation of a images, where it assigns every pixel to specific category. The category is quite versatile and includes machine translation and transcription.
- **Machine Translation:** Input consists of a input sequence of symbols, which is then translated into another language.
- **Transcription:** Is the task to observe a unstructured representation of some data, and transcribe it into textual format. Example an image of handwritten letters, the program is asked to return the handwritten letters in ASCII code for instance.

To evaluate how good the ML algorithm performs, a quantitative measure of performance is needed. Typically the performance is related to the task, such the task itself will carry out the performance measure. For a classification task, the performance of the model is carried out from the accuracy of the model. E.g how good it predicts the correct output of a given examples. For regression problems, the system itself produces an output which the ML algorithm shall mimic. A performance measure of a regression task, is called a loss function. It finds a measure of performance between predicted value and actual value, two common loss function are Mean Square Error (MSE) and Mean Absolute Error (MAE).

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (5)$$

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad (6)$$

where

\hat{y}_i is the predicted value

y_i is the measured value

n is the the number of predictions

But for a regression task, it might not be useful to have a performance measure on already trained data, as the model is specialized to mimic it's trained data well. Therefore one should have separate test dataset, which differs from the training dataset. Such it measure how well the model performs on new data.

A ML algorithm can be categorised into two types of learning, supervised and

unsupervised. They differ in the amount of experience they are permitted to gain during the learning process. For unsupervised, the ML learning algorithm doesn't know anything about its features, e.g. there are no labels. The learning algorithm must learn useful properties of the structure of the dataset in order to perform its task. Supervised learning has the properties of knowing something about where the data comes from, hence data are labeled.

There are some basic terms in ML which are important to learn, they are as follows:

- **Overfitting:** When your model fits exactly against its training data. [14].
- **Underfitting:** Underfitting is when a model can't both model the training data and generalize to new data [14].
- **Generalization:** How well the ML model behaves on new unseen data [14].
- **Regularization:** Different techniques for using a bigger model for reducing generalization. It reduces overfitting and can also lead to faster optimization of the model and better overall performance [15]. These techniques can be "Earlystopping", "Dropout" or a regularization term in cost function.

2.9.1 Feedforward Neural Networks

Artificial neural networks (nets) are a type of machine learning model influenced by research on mammalian central nervous system [16]. A network is made up of numerous layers of connected neurons [16], as illustrated in fig. 16.

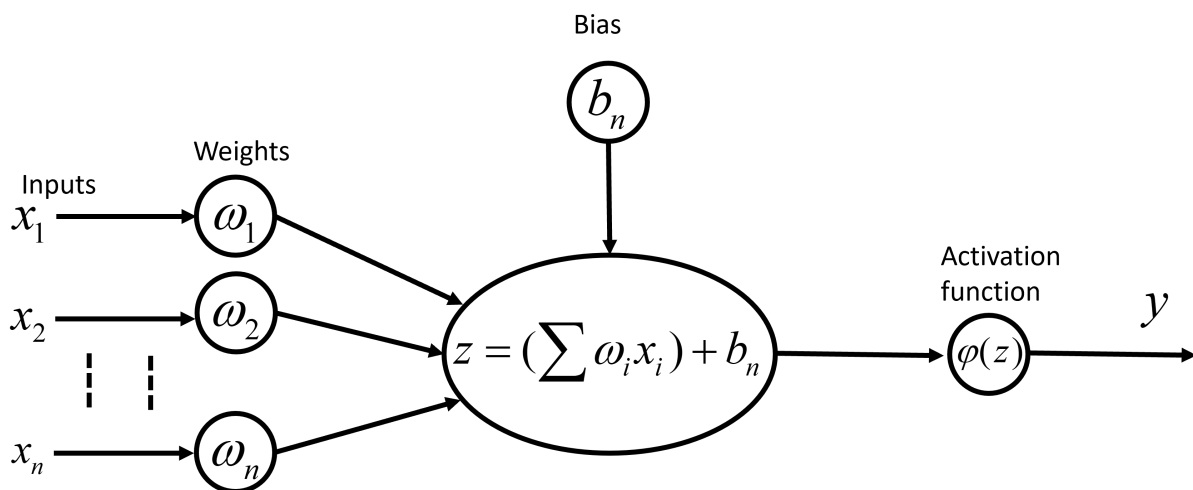


Figure 15: How a neuron looks like mathematically

An artificial neuron, is a mathematically representation based on a model of biological neurons. Each neuron receives inputs (features), weights them individually with a bias, and sums them via a linear or nonlinear activation function [17].

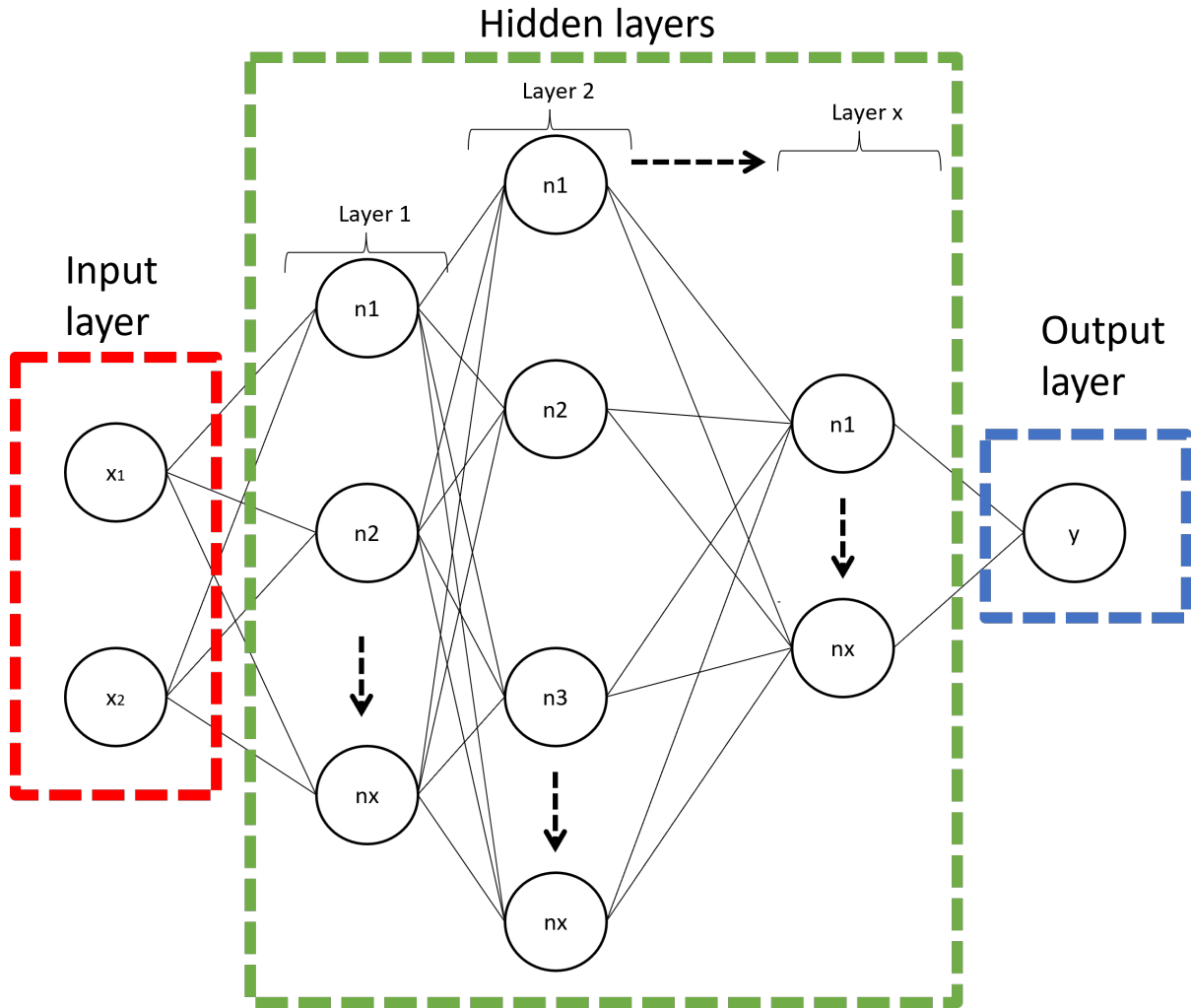


Figure 16: Illustration of a Feedforward Neural Network

There exist two types of activation functions, linear and nonlinear. For feed forward neural networks typically used function are nonlinear functions. The default nonlinear activation function is Rectified Linear Unit (ReLU)[18]. The activation function is widely used, for it's ability to make a nonlinear function out of an output of a linear transformation [18]. Equation (7) describes the ReLU function mathematically. The output is it's input, if the input is greater or equal to zero, else if the input is less than zero, it will output zero.

$$\psi(Z) = \begin{cases} 0, & \text{for } z < 0 \\ z, & \text{for } z \geq 0 \end{cases} \quad (7)$$

2.9.2 Recurrent Neural Network

A Recurrent Neural Network (RNN) is a sophisticated deep learning model that has shown amazing achievements during the previous five years. It uses a sophisticated memory-based architecture to generate predictions on sequential data [19]. Unlike Feedforward Neural Network (FFNN)'s, where information travels in a single path from layer to layer, RNN's feed the output from the previous time stamp together with the input from the current time stamp into the RNN cell, influencing the current state of the model [20]. Equation (8) explains the mathematics behind a single RNN cell.

$$h_t = \tanh(W[h_{t-1}, x_t] + b) \quad (8)$$

W is the weight matrix, b is the bias matrix, h_t and h_{t-1} are hidden state at current and previous time-step. The \tanh scales the cell to fall between -1 to $+1$. For RNN cells, the most used activation function is the sigmoid function. A neural network containing RNN cells, can be difficult when sequence input is too large, this is due to vanishing and exploding gradients when trained by back-propagation[20].

2.9.3 LSTM

To overcome the vanishing and exploding gradients problem of RNN cells, Hochreiter and Schmidhuber [21] proposed Long short-term memory (LSTM). A LSTM cell consists of forget gate, output gate, input gate and update gates. The forget gate determines what to forget from prior memory units, the input gate determines what the neuron will accept, update cell updates the cell, and the output gate is in charge of generating new long-term memories. These four ingredients make sure that it allows for accepting long-term memory, short-term memory, input sequence, and it generates new long-term memory, short-term memory and output sequence at a given time.

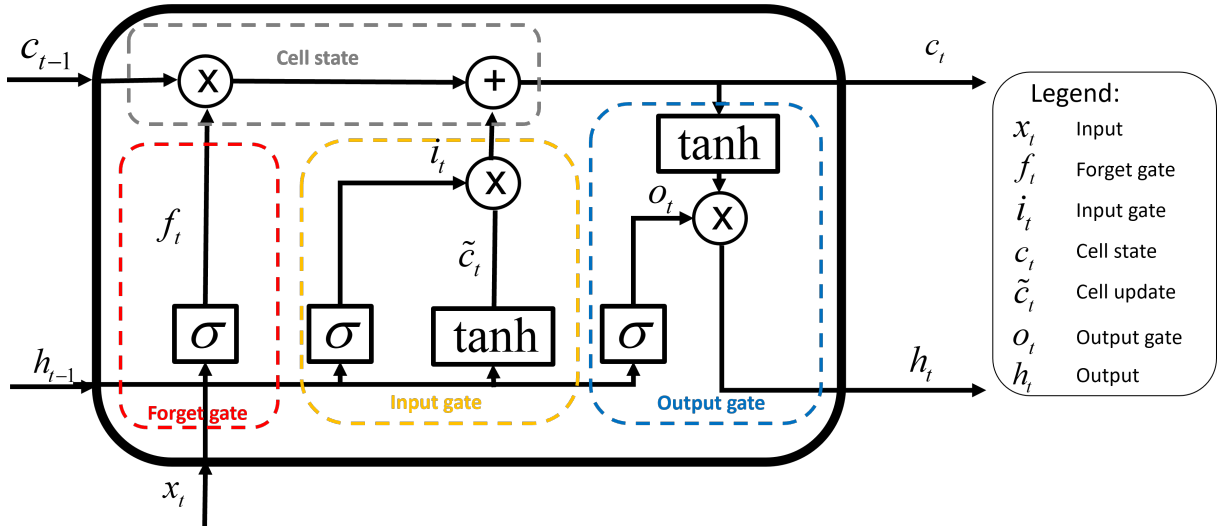


Figure 17: LSTM cell structure

2.9.4 Training loop

All neural networks must be trained in order to be a machine learning algorithm. The neural network must be learned such that the weights and biases are adjusted. For a regression problem the weights and biases are adjusted, such that it maps a function $y = f(x)$ to reassemble some measured values as close as possible. The adjustment (learning) of these weights are done through a optimization problem [18]. A deep neural network has the nonlinear property, meaning training it with a loss function causes it to be nonconvex. Such training of them is done with iterative, gradient-based optimizers that drive the cost function to a low number. For a nonconvex loss function there are no guarantee for convergence, and is sensitive for initial values on weights and biases.

Therefore a good measure is to initialize weights to a small value, and biases to zero [18]. To compute the cost function, the MSE is widely used for regression problems. A basic batch gradient descent algorithm are as followed, first define a cost function ($J(\theta)$). Then loop through each batches (B) in the dataset, forwards pass to compute the cost function as followed:

$$J(\theta) = \frac{1}{|B|} \sum_{i \in B} (y_i - f_{\theta}(x_i))^2 \quad (9)$$

A backward pass to compute the gradient:

$$\frac{\partial J}{\partial \theta} \quad (10)$$

update the parameters with:

$$\theta \leftarrow \theta - \alpha \frac{\partial J}{\partial \theta} \quad (11)$$

This is done over several epochs, an epoch is one pass through with the whole dataset.

2.9.5 TensorFlow

TensorFlow is an open-source python library, for use in numerical computation, to make machine learning models fast and easy. The framework is created by Google Brain team, it uses python as a front-end API for building applications, while executing those applications in high-performance C++ [22]. TensorFlow's most important feature for machine learning development is abstraction. The developer may focus on the overarching logic of the program rather than the small details of developing algorithms or finding out correct methods to hook the result of one function to the input of another. TensorFlow handles all this details behind the hood. While you still can abstract a lot away, there is still flexibility for the developer to dive deeper, for instance customizing your own training loop [23].

3 Sequence

First task of the thesis is to implement a sequence. The process to control is as described in section 2.7 an oil well, but limits to a SISO system which controls the choke and oil rate. In SEPTIC notation, the choke is a MV while the oil rate is a CV. The sequence takes care of starting up the well, regulate it to setpoint, shut the well down and a well off mode. The sequence is converted into a state machine, with different

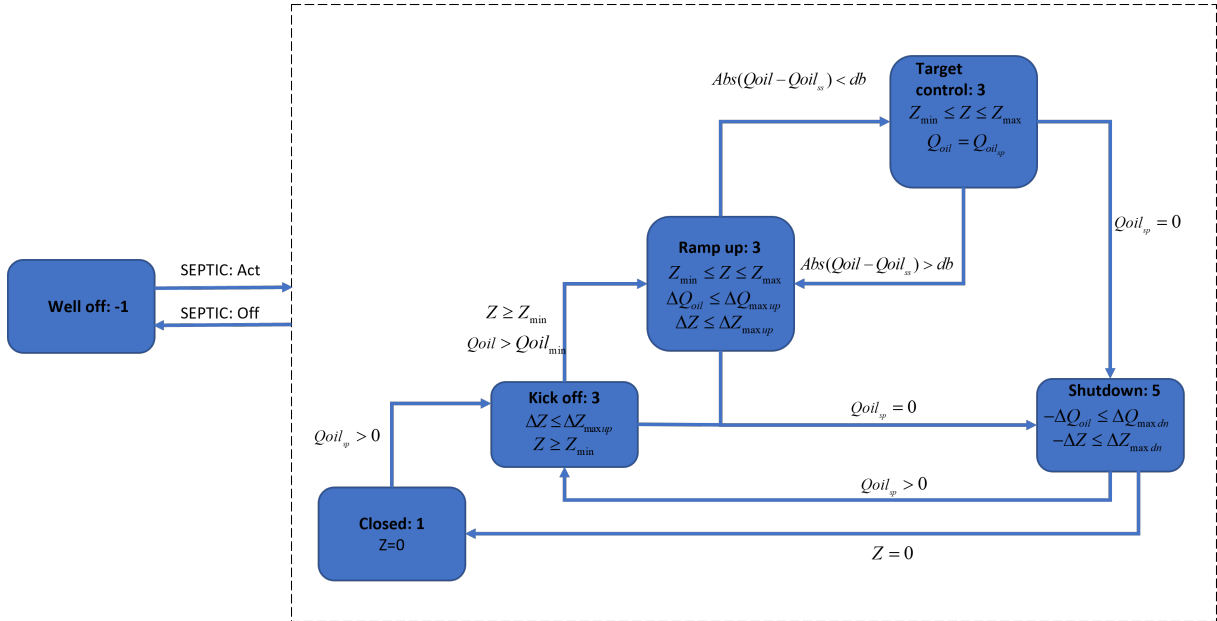


Figure 18: Illustration of the sequence

“Wellstates” which it can be in. Every states illustrated in fig. 18 has a number which represent the state. At all time it can only be into one state at a time. It can only go between the states described in fig. 18, but as an exception it can jump to and from the manual mode “Well Off” state and jump back into any of the states back again, illustrated with dotted lines. The well off state are always active if either the mode of the MPC application are set to “off” or one of the CV “OilRate” or MV “Choke” mode are set to “off”. The transition requirement to change from one state to another are illustrated with arrows with description of the transition criteria. Inside each state there are some system dynamics criteria which must be fulfilled, and configuration of these criterion’s must be tested with pytest. For properly testing how system dynamics can be used with the TDD methodology, some tests must be written before implementation in SEPTIC configuration file. Here it includes both testing of functionality and testing of dynamic of the system.

3.1 Sequence implementation

In accordance to the TDD methodology, one must first write the smallest unit test and make sure it fails, then write code such that the unit test pass. The way to go is to first write a unit test which includes only one state with it’s transitions well state criteria.

And extend further with more tests, and when more functionality are added, the previous tests should/can fail, as new states are added. Such there are loop, where one must always check that previous tests passes. Figure 19 illustrates how to include new

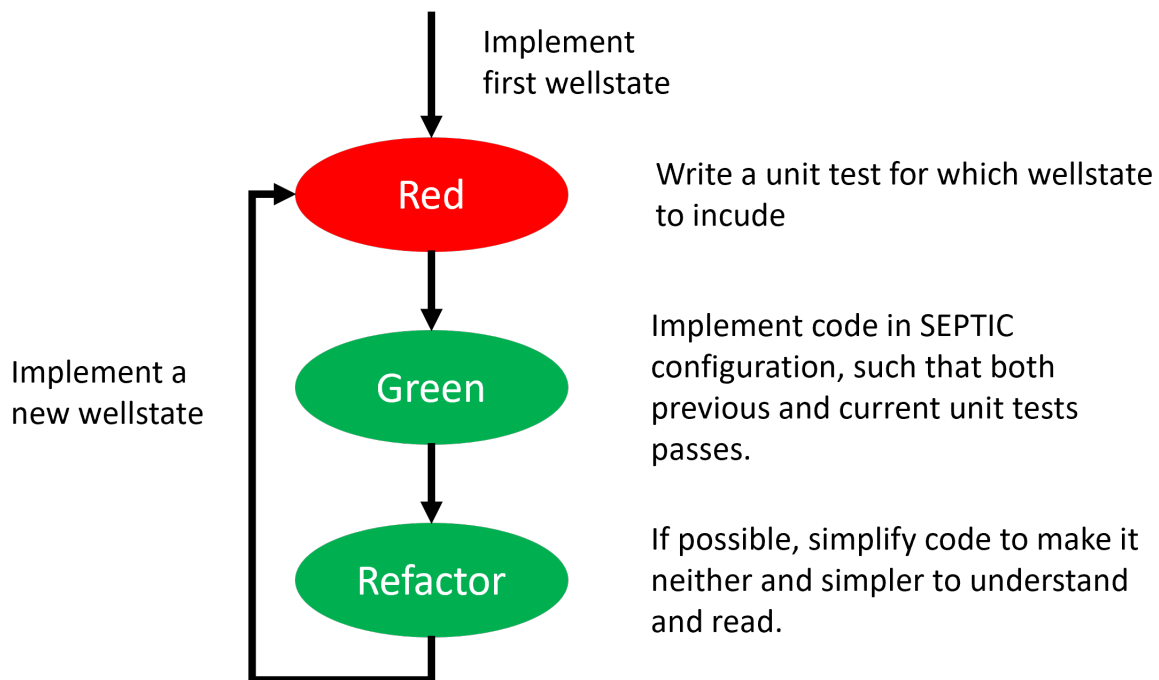


Figure 19: The methodology for sequence implementation

functionality in SEPTIC with TDD. The first loop through, is to write a unit test for the well off state. Then implement the well off state in SEPTIC configuration file such it passes the unit test. The next iteration, is to write a unit test for the well closed state. Then again implement the functionality in SEPTIC configuration file. But it's important to be aware of that the new functionality can interfere with the previous implemented well off stage, therefore a adaptation in the well off state may needed, such that both the unit tests of well off state and well closed state passes. Therefore for each iteration it's important to check previous unit tests such that new functionality doesn't interfere with previous implemented functionality.

To not to bore with too much details of each iterations, a demonstration of the first three iterations are shown. Then you will be showed all of the unit tests and the final SEPTIC configuration file. The rest of the well states are implemented with the same principle, therefore too much details around how they are implemented are not necessary.

3.1.1 Well Off

Listing 3 is a unit test created before any implementation in SEPTIC configuration file are done. To summarize, it check that if both the choke and MPC application are in "off" mode, and check if the "WellState" is set to -1 .

Listing 3: Well Off unit test

```

def test_WellOff(opc, rui, director, director2):
    WellOff=-1
    # set mpc to "off" mode
    opc["Allowactive"]=0
    # set Choke to "off" mode
    rui.appl["SingleWell"]["Choke"].Mode=0
    # step to update values in SEPTIC
    director.step(1)
    # Check that the wellstate is in "welloff" state
    assert rui.appl["StartUpCalc"]["WellState"].Meas==WellOff

```

The implementation of the well off state, are simple. It includes a “DmmyAppl” called “StartUpCalc”, two variables which is calculated in a “CalcPvr” called “TransWellClosed” and “WellState” and a constant which specify which number the well off state represent (−1). “TranWellOff” is the variable specifying the condition of transition to which the “WellState” shall change to “WellOff” state. The calculation of “TransWellClosed” is done through a built in “getfinalstatus()” function. It will get the final status of the choke, it is 3 if both the choke and MPC application are set to “Active”. Such the well will be off if the “getfinalstatus(choke)” is not 3, implied with the “not” function surrounding the “getfinalstatus()”. At last it will update the “WellState” variable with “−1” if the “TransWellOff” is active.

Listing 4: Well Off calc

```

DmmyAppl:    StartUpCalc
    Text1= "Calc to start up well"
    Text2= ""
    Nstep= 1
    PlotMax= 25
    MasterOn= ON
    DesMode= ACTIVE

Evr: TransWellClosed Meas= 0
Evr: WellOffState Meas= -1
Evr: WellState Meas= 0

CalcModl:    Calc
    Text1= ""
    Text2= ""

    CalcPvr:    TransWellOff
    Text1= ""
    Text2= ""
    Alg= "not(getfinalstatus(Choke)==3)"

    CalcPvr:    WellState
    Text1= ""

```



```
Text2= ""
Alg= " TransWellOff*WellOffState"
```

3.1.2 Well Closed

The second state to implement in the sequence is the “Well Closed” state. As earlier, a unit test to test the “Well Closed” state are written. Listing 5 is the unit test, it should be quite self-explanatory. First obtain a stable flow, with some initial value for the choke and gaslift rate. Then close the well carefully by slowly decreasing the choke to zero. Check if the choke is indeed closed and there is now flow going through the well. As the well is still in “WellOff” mode, check that the well state isn’t changed when the well is closed. But when activating it, the wellstate should change from “WellOff” to “WellClosed” state, as included at the bottom of listing 5.

Listing 5: Well Closed unit test

```
def test_WellClosed(opc, rui, director, director2):
    WellClosed=1

    # Set inital value to choke and gasliftrate
    rui.appl["SingleWell"]["Choke"].Deas=20
    rui.appl["SingleWell"]["GasLiftRate"].Deas=10000

    #obtain stable flow
    director.step(800)

    # check that the well is open
    assert rui.appl["SingleWell"]["OilRate"].Meas>0
    assert rui.appl["SingleWell"]["Choke"].Meas>0

    # Close the well down
    for x in range(21):
        rui.appl["SingleWell"]["Choke"].Deas=20-x #Shut the well down
        director.step(25)

    # step sufficient amount of time such that the system is stable
    director.step(125)

    # check that well is closed
    assert rui.appl["SingleWell"]["OilRate"].Meas<0.5
    assert rui.appl["SingleWell"]["Choke"].Meas<0.5
    assert rui.appl["StartUpCalc"]["WellState"].Meas==WellOff

    # Set MPC and choke to active and setpoint to 0
    opc["Allowactive"]=1
    rui.appl["SingleWell"]["Choke"].Mode=3
    rui.appl["SingleWell"]["OilRate"].SetPnt=0
    director.step(10)
```

```
# Check wellstate is in closed state
assert rui.appl["StartUpCalc"]["WellState"].Meas==WellClosed
```

The configuration which includes the “Well Closed” state, is shown in listing 6. Same as the previous, it includes a transition state “TransWellClosed”, which check if the choke is closed and if there are roughly no flow of oil trough the well. A constant which includes what number the state are included as “WellOffState”, in addition to calculation to check if the well is closed (“IsWellClosed variable”). The “IsWellClosed” check if there are no transition to “WellOff” state, and sets the “IsWellClosed” state if it’s transition are active or if the “WellState” already are in “WellClosed” state. Such creating similar to a SR latch, in combination with “WellState” calculation.

The only which can reset the “WellState” from being “WellClosed” is the “TransWellOff” which suppress the set condition in the “and” gates in “IsWellClosed”. This is necessary, since both the transition variable may be active, but only one state is allowed to be active, such the “WellOff” will always be active if it’s transition is active. But the “IsWellClosed” are only allowed to be active if the well state is not in “WellOff” state and “TransWellClosed” is true.

Listing 6: Well Closed calc

```
DmmyAppl:    StartUpCalc
  Text1= "Calc to start up well"
  Text2= ""
  Nstep= 1
  PlotMax= 25
  MasterOn= ON
  DesMode= ACTIVE

  Evr: TransWellClosed Meas= 0
  Evr: WellOffState Meas= -1

  Evr: WellState Meas= 0

  Evr: WellClosedState Meas= 1
  Evr: IsWellClosed
  Evr: TransWellOff Meas= 0

  CalcModl:    Calc
    Text1= ""
    Text2= ""

//----- Transitions-----
  CalcPvr:    TransWellOff
    Text1= ""
    Text2= ""
    Alg= "not(getfinalstatus(Choke)==3)"
```

```

    CalcPvr:      TransWellClosed
        Text1= ""
        Text2= ""
        Alg= "and(OilRate<0.5,Choke<0.5)"

// -----Calculation of WellState-----

    CalcPvr:      IsWellClosed
        Text1= ""
        Text2= ""
        Alg= "or(and(TransWellClosed, not(TransWellOff)),
              and(WellState==WellClosedState,not(TransWellOff)))"

    CalcPvr:      WellState
        Text1= ""
        Text2= ""
        Alg= " TransWellOff*WellOffState+IsWellClosed*WellClosedState"

```

3.1.3 Well Kick Off

The well kick off state, differs from the previously implemented states, as it includes system dynamics testing. Listing 7 is the unit test of the “Kick Off” state, the test differ from previous unit tests as it includes uncertainty of how many steps to simulate. In this example, the test shall includes to check if the choke goes above the minimum choke opening of Z_{min} , but how many steps needed in order for the choke to be opened to Z_{min} can vary. Such a function that handles this uncertainty was made. It is called *step_until*, it can be later included in the director class, but as for now it is made in a separate class, and sent to the unit test through the conftest file as an object called “director2”, not interfering with the director class.

The function takes the parameter *appl*, *xvr*, *criteria*, *comperator* and the optional parameters *value="Meas"* and *maxsteps=1000*. The first two parameters (*appl* and *xvr*) specify path to the XV object in SEPTIC, in addition to the value parameter which specify which member of the XV it shall use, default value is “meas”, but can be changed to for example “Mode”. The *criteria* and *comperator* parameter specifies the criteria which the value from SEPTIC shall be compared with. The *comperator* parameter is inherited from the operator module, such the *comperator* can be set to any *comperator* as wanted. To specify the *comperator* to equal, simply set the *comperator* to *operator.eq*, or to set it to greater or equal, set it to *operator.ge*. The *maxstep* parameter specify the maximum number of steps, when the condition of the step until is not met. The function returns number of steps stepped, the code of this class are added in appendix A.1.

To test the dynamics of the system, first must the system be simulated, and then

extract the dataset of the simulation. There are three ways of extracting these values from SEPTIC.

1. Convert a auto generate .dta file from SEPTIC to a readable format for python.
2. For each step, read simulation value and add it to an array.
3. As simulation values are stored in a ring-buffer, simulate the amount of time wanted, then retrieve simulation by accessing the ring buffer.

A sensible choice here would be item 3, as access and editing of data is easy to do while the unit test is running. Listing 7 is a unit test for “WellKickOff” state. In addition to check if the “WellState” is in right state with the right transition criteria, it checks the rate of change in choke. It simulate the system with the `step_until()` function, to reach the minimum choke opening value of 10% .Then accessing the ring buffer of the simulated choke opening values, and maps it into a list . Then check if the maximum value of the numerically computed derivative of the choke opening, is less than maximum allowed rate of change in choke.

Listing 7: Well Kick Off unit test

```
def test_WellKickOff(opc, rui, director, director2):
    # Specify minimum choke opening and rate of change of Choke opening
    Zmin=10
    deltaZmaxUp=0.1

    # Set OilRate setpoint to 290
    rui.appl["SingleWell"]["OilRate"].SetPnt=290
    # Step until Wellstate equals WellKickOffState
    N_step=director2.step_until("StartUpCalc", "WellState", WellKickOff,
        comparator=operator.eq)
    #Check that Wellstate are indeed in WellKickOff state
    assert rui.appl["StartUpCalc"]["WellState"].Meas==WellKickOff

    # Step until the choke reaches it's minimum choke value of Zmin
    N_step=director2.step_until("SingleWell", "Choke", Zmin,
        comparator=operator.ge)
    # Check if the choke has reached it's minimum choke value of Zmin
    assert rui.appl["SingleWell"]["Choke"].Meas>=Zmin

    # extract the simulated choke values from septic
    Z=list(map(itemgetter(0), rui.appl["SingleWell"]["Choke"].Meas[-N_step:]))
    # compute the rate of change of the choke numerically
    deltaZ=np.gradient(Z)
    #Check that the rate of change isn't above maximum allowed rate of change
    assert round(max(deltaZ),2)<=deltaZmaxUp
```

Appendix B.1 is the result of the configuration file for the “WellKickOff” state. It follows same principle as previously implemented states, with a transition and a “IsWellKickOff” state. Previous implementation has been modified, in order for the

sequence to progress to next state. It includes reset of previous state (not(TransKickOff) in IsWellClosed), when transition of the next state is active. There are also logic, setting boundaries for minimum and maximum choke opening.

The unit test listing 7 includes testing of tuning parameters in SEPTIC, more specific the maximum allowed rate of change of choke opening. To specify the maximum allowed rate of change in SEPTIC, a tuning parameter in the configuration file must be set. More specific the “MaxUp” in the choke must be set to “0.1”. And as a result the three unit tests all together passes with the modification done. Indicating a successfully implementation of a sequence with system dynamics tests included.

3.1.4 Full Sequence Implementation

The rest of the sequence is implemented with the same principle as the three other well states. First a unit test of the well state is written, then implement in SEPTIC configuration file. For each implementation, modification of previous states must be slightly modified for it to incorporate the new state. Appendix A.3 is all of the unit test for the sequence, and appendix B.2 is the SEPTIC configuration calc which implements the sequence.

The “WellRampUp” state is a very interesting state, the ΔQ_{maxup} condition in the state, requires the unit test to test system dynamics. As seen in the implementation of the “WellKickOff” state, a tuning variable had to be modified such it met requirement of maximum rate of change in choke. But the maximum allowed rate of change in oil rate, hasn’t got any direct variable setting it. It must be tuned indirectly, by tuning parameters in the choke and oil rate. The tuning parameters is fulf, movePnlty and maxUp for MV’s and fulf for CV’s, in addition to priority number for MIMO system.

3.2 Simulation

Figures 20 and 21 is the simulation of the sequence. The sequence is simulated with the sequence unit tests attached in appendix A.3. Figure 20 shows how the oil rate and choke opening plays out throughout the different sequence states. Figure 21 plots the state the sequence is in. It shows, the sequence is successfully implemented. It goes through the different states as specified in fig. 18. Not all transitions is tested in this sequence, transition to and from “WellOff” is not tested to all states, only to “WellClosed”. Another state which almost every state can go to is the “WellClosed”, the unit tests only tests going from “WellShutdown” but misses to check if it can go from “WellKickOff” and “WellRampUp” to “WellShutdown”. Such the test is a bit deficient in terms for checking all transitions.

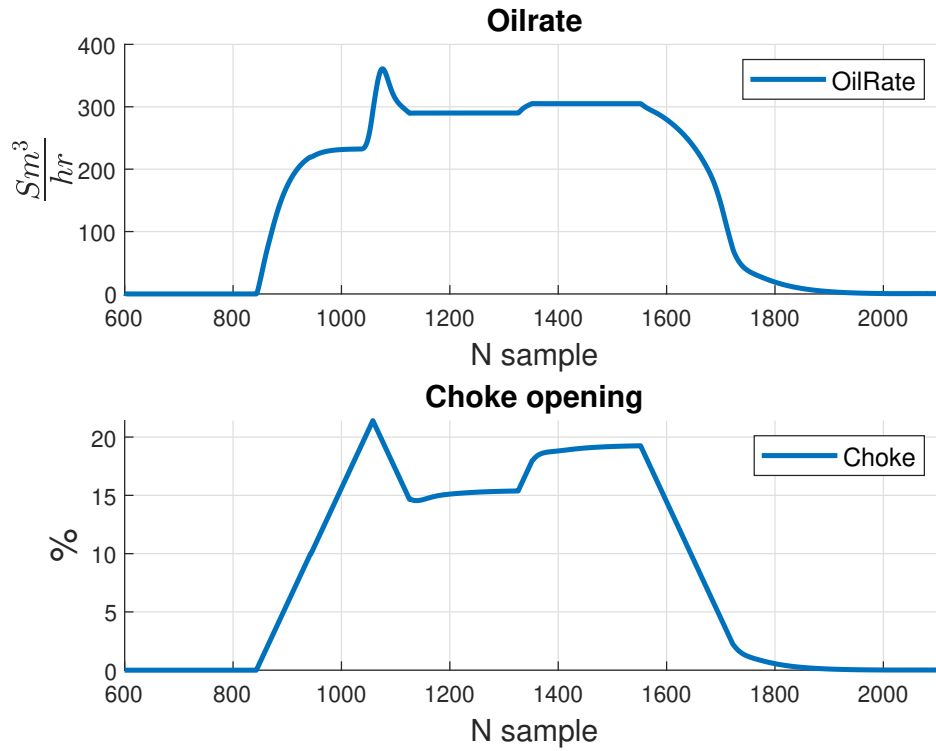


Figure 20: Plot of Choke and OilRate

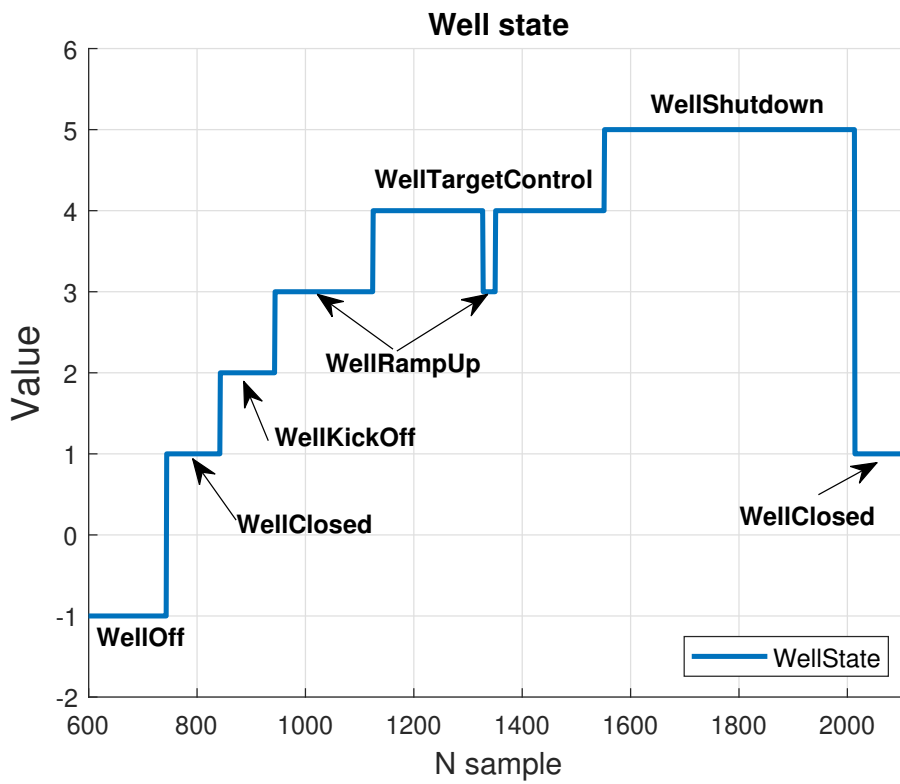


Figure 21: Plot of the wellstate

4 Machine Learning of System Dynamics

The second thing to look into was learning of system dynamics with use of machine learning. The main goal is not to learn the system dynamics flawless, but learn the steady-state gain between the oil rate and choke opening. Further a gain scheduling calc shall be implemented, using the ML model as basis for calculating a gain function with choke as input parameter.

4.1 Learning problem

The very first ingredient of a ML algorithm is to define the learning problem. The ML algorithm's task, is to learn the steady-state oil rate from the choke opening. This translate to a regression problem mapping the choke opening to a oil rate. The model shall generate a function $y = f(x)$. The input of the model is choke opening and output is oil rate. A naturally performance measure is therefore the real measured oil rate. The training experience is gained from a simulation of the oil well process, containing different choke opening values from 0% – 100% and the measured output of oil rate.

4.2 Dataset

In a ML algorithm, it is really important to find a good dataset to train and test on. The training set must be well defined, and have sufficient amount of training sample. At first, a training set consisting of stepping the choke 5% from 0% to 100% was made. However, when the dataset was evaluated, it found it somewhat non-linear in the region 0% – 15%. Including this region in the training set, would give bad generalization of the rest of the working area when the model is trained. Therefore it was made a decision to not include this region in the training set, and only focus on 15% – 100%.

Addition to this, a initialisation of the well is necessary. Because stepping the well up and down 15% – 100%, will give different steady-state behaviour for the same choke opening when going down compared to going up, which is not favourable and quite unpredictable. Therefore the choke must be initialized first by stepping 15% to 100% and back to 15%. Then perform same stepping procedure again which the training set is contained. This ensures reliability for the model such that same choke opening will produce the same steady-state behaviour in oil rate everywhere.

The test set is made in order to check how good representation the model are on any given similar data. A typically pitfall in ML algorithm is overfitting the model on the training set, e.g be too specialized on training set. Such when any new data are given to the model, it would perform bad, but perform very well on the training set. A common strategy is to extract 10% of the training set to the test set. But since the training set is quite uniform in terms of stepping 5% each time. A test set including different stepping length near 5% is favourable, for looking how well the model is generalized into other stepping lengths.

4.2.1 Training set

To save time, the same test environment as section 3 is used to generate the training set. A “unit test” which doesn’t check any thing is created with the wanted simulation. Listing 8 is the code for creating the training set, and fig. 22 is plot of the training set. The choke goes from 15% to 100% and back, with stepping length of 5% and sufficient amount of simulation time to reach steady-state.

Listing 8: Training set python code

```
def test_step15to100(opc, director, rui):
    choke=15
    time=200

    rui.appl["SingleWell"]["Choke"].Deas=0
    director.step(500)
    rui.appl["SingleWell"]["Choke"].Deas=choke
    director.step(350)

    #initialise the well
    #step 15 to 100
    for choke in range(15,105,5):
        rui.appl["SingleWell"]["Choke"].Deas=choke
        director.step(time)
    # step 100 to 15
    for choke in range(100,10,-5):
        rui.appl["SingleWell"]["Choke"].Deas=choke
        director.step(time)

    # create the dataset
    #step 15 to 100
    for choke in range(15,105,5):
        rui.appl["SingleWell"]["Choke"].Deas=choke
        director.step(time)
    # step 100 to 15
    for choke in range(100,10,-5):
        rui.appl["SingleWell"]["Choke"].Deas=choke
        director.step(time)
```

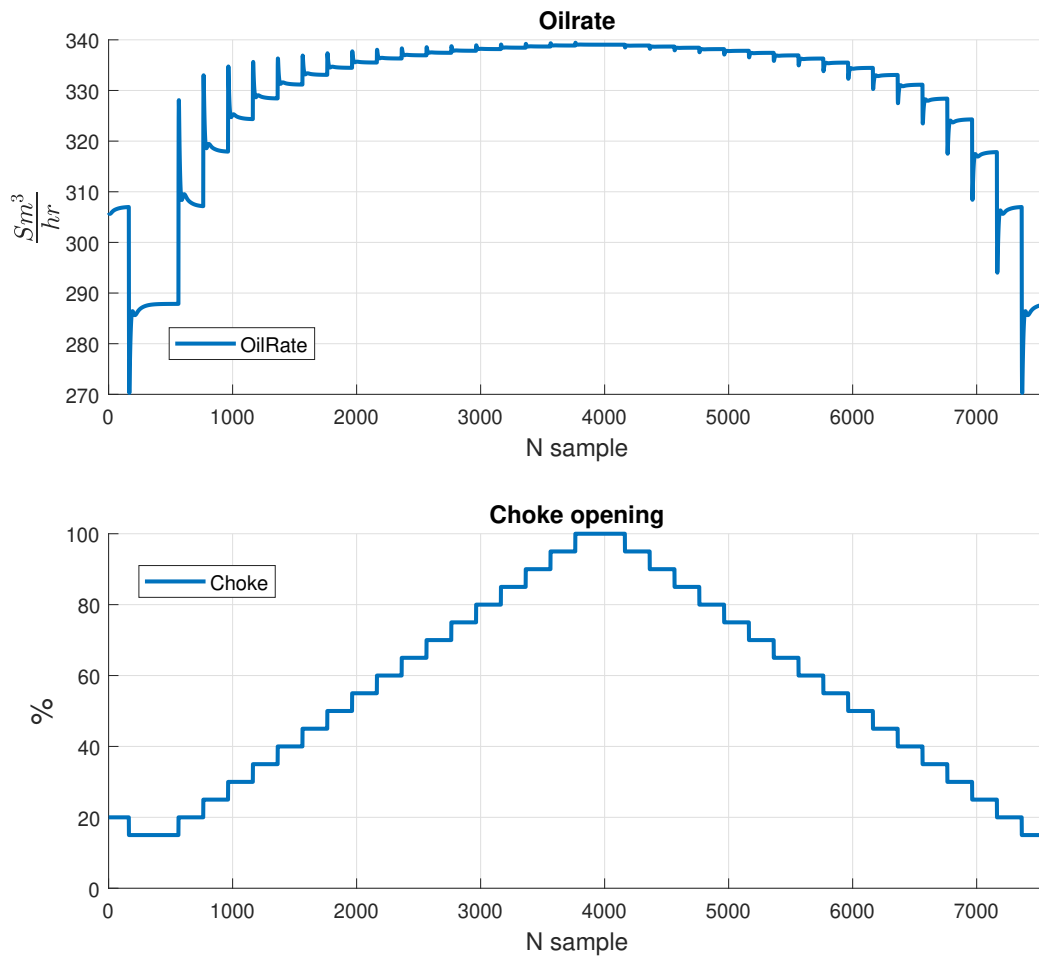



Figure 22: Training dataset

4.2.2 Test set

The test set is generated in the same matter as the training set, with a unit test not checking anything. Appendix A.4 is the python file generating the data, while fig. 23 is plotting of the test set. It includes stepping in the region of 5%, to not go too far from the training set, such the test set has roughly same distribution.

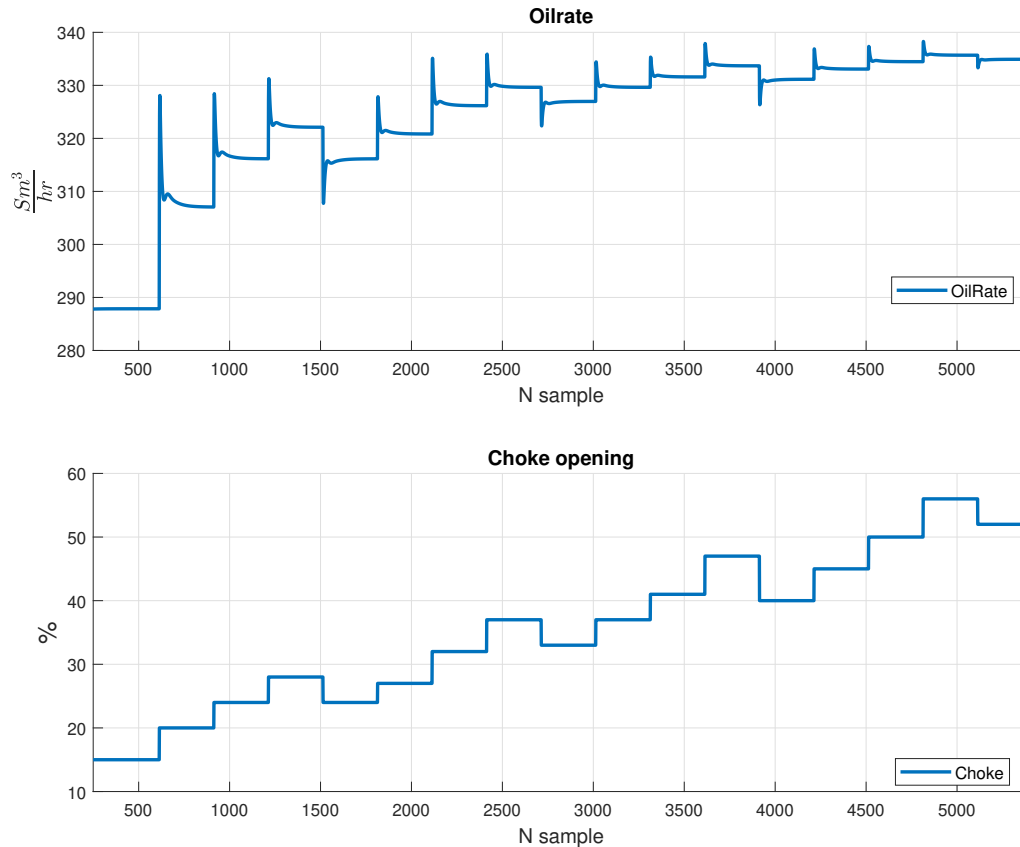


Figure 23: Test dataset

4.2.3 Convert data

When the test and training set simulation is done, a way of accessing the data is necessary to create a ML model. SEPTIC creates a auto-generated .dta file, this file contains simulation of almost every state in SEPTIC. But the format is limited and it can't be directly read in python, and must be converted to a readable format. In data science a common file format is comma-separated values (CSV), and thereby chosen to be converted to. Appendix A.5 is the python script which converts the data to .csv file. The approach is to copy the auto-generated .dta file, save it as a .txt file. Set the .txt path in the "Convert_Data.py" script and convert it. The script has also the ability to specify the number of sample to skip in the beginning.

4.2.4 Data preprocessing

As LSTM neural networks demands sequential data, some data preprocessing of the data must be done first. Conversion is done by defining a sequence length, then for each data point in the dataset, the sequence contains the next data of the length of sequence length. For better understanding let's look at a dataset containing following, with

sequence length of 3:

$$[2, 4, 6, 8, 10] \quad (12)$$

The sequence's of eq. (12) will then be:

$$[2, 4, 6] \quad (13)$$

$$[4, 6, 8] \quad (14)$$

$$[6, 8, 10] \quad (15)$$

Two types of “create_sequence” is made, one for creating sequential data for only features, and one for creating sequential data for both features and output. Listing 9 is the python code which creates the sequential data.

Listing 9: Data sequence conversion code

```
import numpy as np
def create_sequence(Xd, Yd, sequence_length=250):
    X=[]
    Y=[]
    for i in range(sequence_length, len(Xd)):
        X.append(Xd[i-sequence_length:i,:])
        Y.append(Yd[i])
    return np.array(X), np.array(Y)

def create_single_sequence(Xd, sequence_length=250):
    X=[]
    for i in range(sequence_length, len(Xd)):
        X.append(Xd[i-sequence_length:i,:])
    return np.array(X)
```

Since RNN's are sensitive to time series data fluctuations and catching patterns in time series data, the data must be normalized before being fed to the neural network. To normalize the data, a “MinMaxScaler” is created to scale the values between -1 and 1 . Equation (16) scales the value x with a max value of x_{max} and minimum value of x_{min} , the normalized scaled value is equal to x_n , eq. (17) descales the values other way around.

$$x_n = 2 \frac{x - x_{min}}{x_{max} - x_{min}} - 1 \quad (16)$$

$$x = \frac{(x_n + 1)(x_{max} - x_{min})}{2} + x_{min} \quad (17)$$

Listing 10: MinMaxScaler python code

```
import numpy as np
def MinMaxScale(data, min=0, max=100):
    data_scaled=[]
```

```

for value in data:
    x=2*((value-min)/(max-min))-1
    data_scaled.append(x)
data_scaled=np.array(data_scaled)
return data_scaled.reshape(len(data_scaled),1)

def MinMaxDeScale(data, min=0, max=100):
    data_descaled=[]
    for value in data:
        x=0.5*((value+1)*(max-min))+min
        data_descaled.append(x)
    data_descaled=np.array(data_descaled)
    return data_descaled.reshape(len(data_descaled),1)

```

When all functionality which scales and create sequences out of the data is finished, one must implemented it with the data made earlier. Listing 11 is does this, before the code is executed, a import of different packages and functions are executed attached in appendix C.1.

Listing 11: Read training set, scale it and create sequences out of it

```

# read training set
train_df = pd.read_csv('TrainingDataset.csv', index_col=False)

# Scale training set
train_x=MinMaxScale(train_df['Choke'], min=0, max=100)
train_y=MinMaxScale(train_df['OilRate'], min=0, max=max(train_df['OilRate']))

# Create sequences out of the scaled training dataset
sequence_length=250
sequence_X, sequence_Y= create_sequence(train_x,train_y,
    sequence_length=sequence_length)

```

4.3 Create and Train a Machine Learning Model

To create the model, one use the python interface to tensorflow named keras. The interface initialize, creates and train a ML effortlessly. Listing 12 is the final model created, but has been changed throughout the training phase. One of the main hyperparameters in ML model is the number of layers and the number of neurons in each layer. There are no exact rule of how many layers or neuron one should have, and the model in listing 12 is carefully adjusted after numerous of trials and errors during training phase. The regularization techniques used are “EarlyStopping” and “Dropout” with a rate of 0.1. “EarlyStopping” stops the training if there are no improvement in loss function, such avoiding to train too much and overfitting the model to the training data. “Dropout” is a technique where random rate of nodes is excluded or ignored in each training stage.

Listing 12: Python code for creating a ML model

```

# input parameters to input layer
timesteps=sequence_X.shape[1]
data_dim=sequence_X.shape[2]
#initialize model
model= Sequential()
# add LSTM with 100 LSTM cells
model.add(LSTM(units=100, return_sequences=True, input_shape=(timesteps,
    data_dim)))
# add dropout with dropout rate 0.1
model.add(Dropout(0.1))

model.add(LSTM(units=50,return_sequences=True))
model.add(Dropout(0.1))

model.add(LSTM(units=50,return_sequences=True))
model.add(Dropout(0.1))

model.add(LSTM(units=50,return_sequences=True))
model.add(Dropout(0.1))

model.add(LSTM(units=100))
model.add(Dropout(0.1))

model.add(Dense(units=1,activation='relu')) # Output layer

```

Some hyperparameters must be configured to train the model. These are the choice of optimizer algorithm, associated learning rate parameters, and choice of loss function. The “Adam” algorithm is chosen as optimizer, as it is very robust with regards of choice of hyperparameters. The parameters can often be set to default and in generally perform well, though learning rate can sometimes differ from default values [18]. The loss function used is the MSE, as it’s the most common for regression problems. The number of epochs chosen is five, the number is chosen after experimental training, and looking on how fast and how many epochs the model needs to be trained for getting sufficient performance on the model. Listing 13 is the code which defines loss function, optimizer and trains the model with the specified hyperparameters. Initialization of the weights is chosen random from a uniform distribution bounded between $\pm\sqrt{\frac{6}{inputs+outputs}}$, and the biases are set to zero. The recurrent weight matrix is initialized as a orthogonal matrix. The output activation function is tanh-function and the recurrent activation function is a sigmoid-function. All of the initialization and activation functions is default values from keras, and is found in the keras API documentation [24].

Listing 13: Python code for training model

```

# optimizer adam
opt='adam'
# loss function mean square error

```

```
lossfun='mse'  
#Configure the model for training with loss function and optimizer algorithm  
model.compile(optimizer=opt, loss=lossfun, metrics=['acc'])  
  
# add earlystopping  
es= EarlyStopping(monitor='loss', mode='min', verbose=1, patience=2)  
#train model with feature sequences generated, 5 epochs and earlystopping  
history=model.fit(sequence_X, sequence_Y, epochs=5, callbacks=[es], verbose=1,  
                  workers=4,use_multiprocessing=True)  
# save model for later use  
model.save('model_scaled.h5')
```

4.4 Evaluation of model

During training it is important to evaluate the model. Evaluation can be done through the quantitative analysis by calculating the MSE value between the model and the prediction made by the model. The MSE value is used in training to evaluate if a training of model is improved in relation to previously trained models. For the final model the MSE value is 8.69. It is also a nice practice to plot the predicted values and measured values, during training and evaluation to see in which parts the model doesn't fit the training data. Figure 24 is the plot of the prediction versus the measured oil rate. The prediction seems to predict steady-state oil rate fairly well, but miss to model the overshoot. But since the model shall be used to predict steady state gain, all dynamics is not necessary to be modelled, the most important is its ability to predict correct steady-state oil rate.

When one is happy with how well the model fits the training set. The model must be evaluated on the test data, to look into how good the model is generalized on new unseen data. Figure 25 is the plot of the predicted and measured oil rate for the test dataset. The model predicts the steady-state oil rate very well, and it looks like the model is generalizing sufficient enough. In matter in fact, the MSE value is 6.74 for the test set, indicating a better score for the test set compared to the training set.

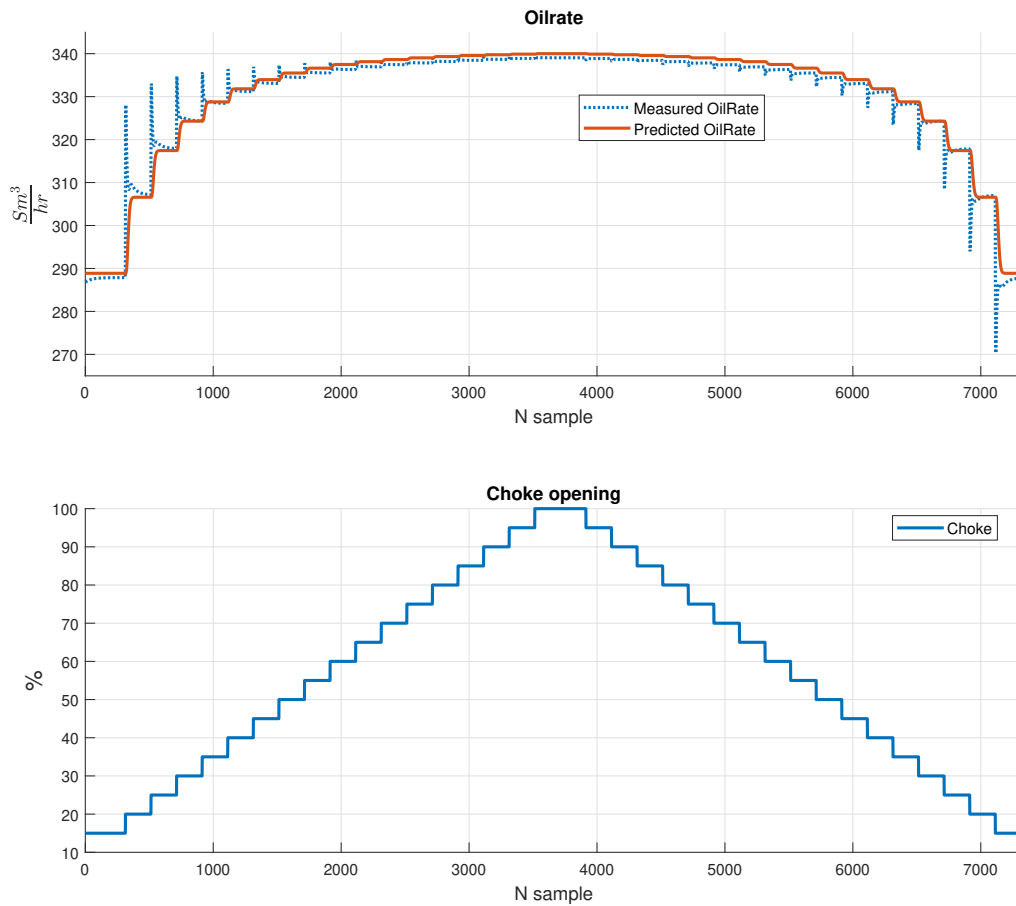


Figure 24: Evaluation plot of training set

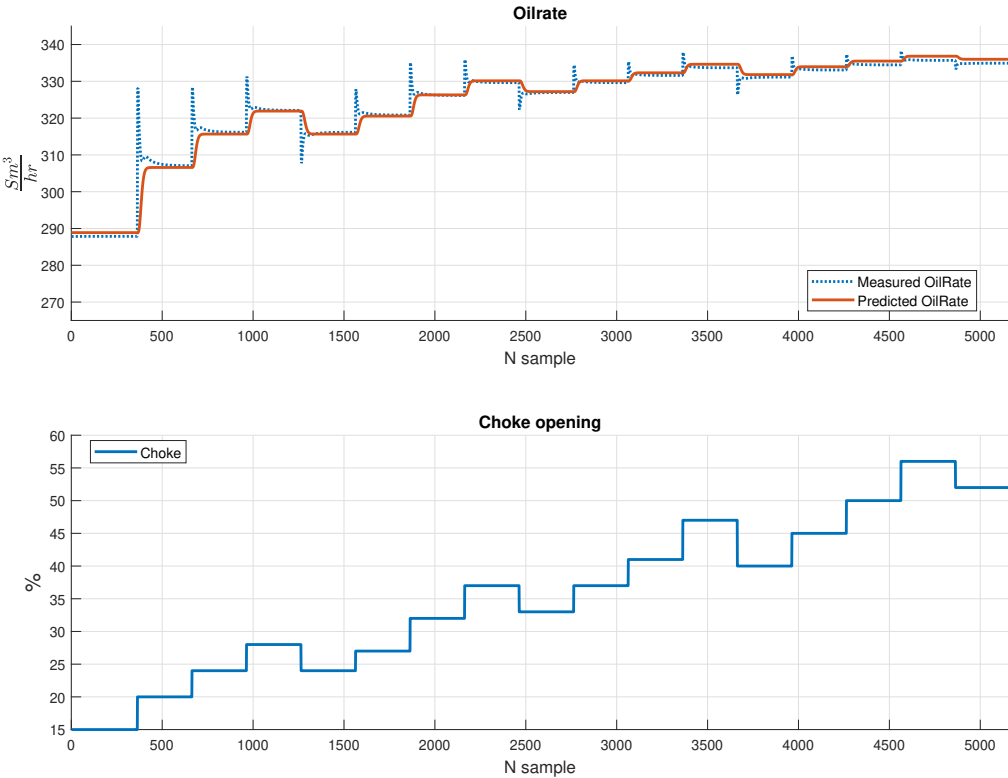


Figure 25: Evaluation plot of test set

4.5 Gain scheduling implementation in SEPTIC

To create a gain scheduler in SEPTIC, one first define a new calc. Then define a gain function, based upon calculated gains from the ML model implemented earlier. The gains are calculated with the following formula:

$$K_{gain} = \frac{\Delta OilRate}{\Delta Choke} \quad (18)$$

To make a gain function, SEPTIC has a function called “intpoltype1” which is linear interpolation between a set of data points. Linear interpolation creates a function with a straight line between the data points, to approximate the value in between.

Generation of these data points is done with a python script, which uses the ML model to approximate the gain. The $\Delta Choke$ in eq. (18) is set to a constant value of 5%. Since the model isn’t learned on choke value lower than 15%, a approximation of the gain between 0% – 15% is done by, setting $\Delta Choke = 15$ and read the oil rate of the ML model at 15% choke opening. Since the oil rate is 0 at 0% choke opening. The gain would be the oil rate at 15% divided by 15. The rest of the gains are calculated with a for loop with step of 5 and goes from 15 – 100. The for loop automatically read $\Delta OilRate$ and divides it by 5 to generates the gain.

Listing 14 is the configuration code in SEPTIC which implements the gain scheduler. There is two EV’s, the “Gain” EV is the one which creates a gain function with the “intpoltype1” function, and “GainScheduler” sets the gain of the model in SEPTIC. The “intpoltype1” takes the “x” argument Choke, pair of X_n, Y_n and return the interpolated gain value based upon the “x” argument. The gains are created between two values of choke opening, the gain is therefore the average over the two choke opening values. Take for instance the gain between 15% – 20% is 3.46, but the pair in the interpolation would be (17.5, 3.46). Figure 26 is a plot of the gain function. The function starts at higher values and grows closer and closer to zero, very similar to a exponential function.

Listing 14: SEPTIC config file for gain scheduling

```
DmmyAppl:      ModelGainCalc
  Text1= "Calc to update gain in experimental model"
  Text2= ""
  Nstep= 1
  PlotMax= 25
  MasterOn= ON
  DesMode= ACTIVE

  Tvr: Apply

  Evr: Gain
  Evr: GainScheduler

  CalcModl:      ModGain
  Text1= ""
```

```

Text2= ""

CalcPvr:      Gain
Text1= ""
Text2= ""
Alg= "intpoltype1(Choke,7.5,19.25,17.5,3.46,
22.5,2.134,27.5,1.35, 32.5,0.886,37.5,0.603,42.5,
0.424,47.5,0.306,52.5,0.226,57.5,
0.170,62.5,0.129,67.5,0.100, 72.5,0.0778,77.5,0.0607,
82.5,0.0475,87.5,0.0371, 92.5,0.0289,97.5,0.0224)"

CalcPvr:      GainScheduler
Text1= ""
Text2= ""
Alg= "modgain(OilRate,Choke,Gain,Apply)"

```

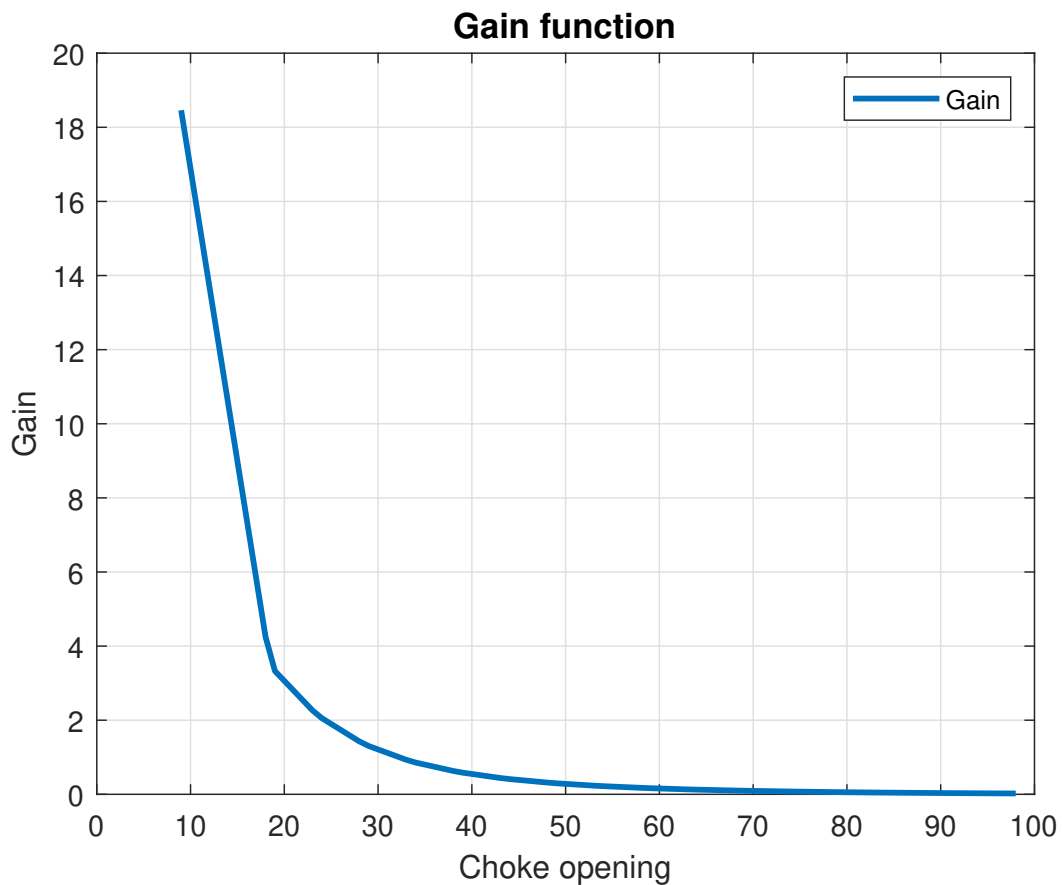


Figure 26: Plot of the gain function

4.6 Simulation on sequence

To test the gain scheduler implemented in section 4.5 to check if the gain scheduler gave any performance improvements. The sequence going from well closed to well target state was simulated. Figure 27 is the plot of the simulation with and without a gain scheduler. The two plots are pretty similar, but the simulation with gain scheduler gave a slighter better performance, reaching the set-point of $335 \frac{Sm^3}{hr}$ quicker than the simulation without a gain scheduler.

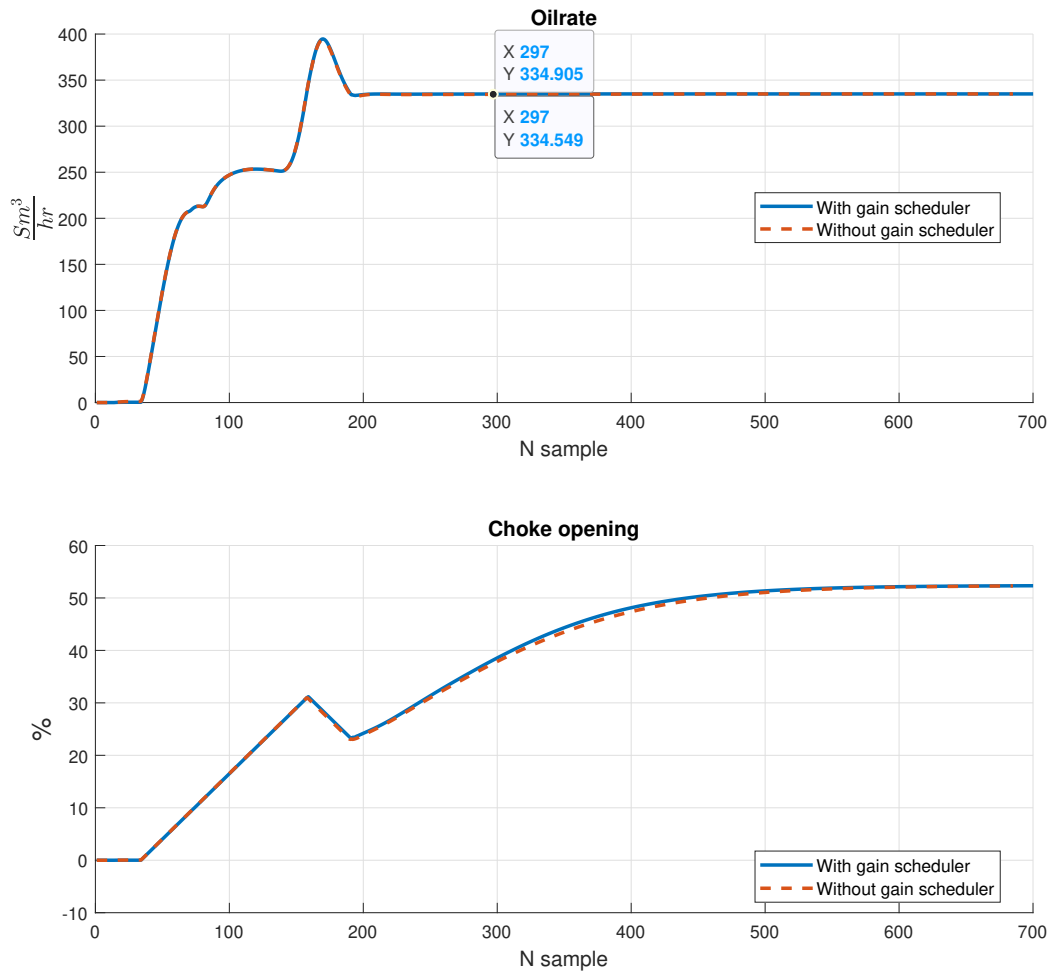


Figure 27: Simulation with and without gain scheduler

5 Discussion

This section discusses the results from sections 3 and 4.

5.1 Sequence Improvement

In sections 3.1 and 3.2 a running sequence starting, running and closing a well was implemented. It was implemented with the use of TDD methodology. The implementation in SEPTIC was quite convoluted to write and understand, with many variables to assign. The implementation included a state machine, where each state was checked by creating similar to a SR latch. The SR latch was created by using confusing amounts of “or” and “and” gates. A much neither way would be if Equinor made a SR latch function in SEPTIC config. The function would lessen the SEPTIC config code, as the transition into the state would set the state, and the transition out of the state resets the state. Implementing such function in SEPTIC would demand small effort, but make huge difference when implementing a sequence, with regards to writing and reading the code. Listing 15 shows how the state originally was implemented, while listing 16 shows how implementation with a potentially SR latch would be. Obviously listing 16 demands much less code to write, and is so much easier to understand.

Listing 15: Code for IsWellClosed calc

```
CalcPvr:      IsWellClosed
              Text1= ""
              Text2= ""
              Alg= "or(and(TransWellClosed, not(TransWellOff), not(TransKickOff)),
                    and(WellState==WellClosedState,not(TransWellOff),
                    not(TransKickOff)))"
```

Listing 16: Code for IsWellClosed calc with a potential SR latch

```
CalcPvr:      IsWellClosed
              Text1= ""
              Text2= ""
              Alg="srlatch(TransWellClosed, or(TransWellOff,TransKickOff))"
```

5.2 TDD of system dynamics

The sequence in section 3.1 was implemented with the use of TDD methodology. The unit tests didn’t just test functionality, it simulated and tested system dynamics in SEPTIC. A key factor for testing system dynamics is to have the *step_until()* function, which gives flexibility for the programmer to extract exact simulation to pytest for further investigate and test the simulation in an array.

As for the result, it was only tested rate of change of choke and oil rate. The maximum rate of change in choke could easily be set by a specific tuning variable in the SEPTIC config. Therefore instead of producing simulation and calculate the derivative of the

choke, a much faster way would be to check through RUI that the “maxup” for choke was set to correct value. But this method would be impossible for the maximum rate of change of oil rate, as oil rate is determined by tuning variables for both choke, oil rate, and for a more common MIMO system gas lift rate and other CV’s influencing the control targets in the MPC.

Further it seems possible to extend the testing capabilities to test overshoot, rise time and peak time and others system response parameters, as these are simple mathematics when the simulation is available. What is left out for further work in regards to TDD of system dynamics, is to extend the MPC to MIMO system, and completely test complex system dynamics parameters such as overshoot and rate of change influencing multiple CV’s and MV’s. Such system would be harder to test and development with TDD, as there is a lot of tuning parameters influencing the response of the system. Because of this, one of the main advantages of TDD namely pin pointing where in the code the test fails disappears, as there might be a couple of possibilities of tuning variables which can affect the system dynamics, making the unit test to pass.

5.3 Machine learning of system dynamics

The ML model made in section 4 managed to learn the steady-state oil rate quite well. The evaluation of the model on the test dataset, showed that the model generalized very well. But even though there are some conceptual flaws with the ML model. The stepping of the training dataset is set to a fixed length 5%, a more random input sequence of the training set might generalized the ML model even better. And the decision to not include 0% – 15% choke opening, because of the highly unpredictable non-linear behaviour of the oil rate, makes the ML model useless for this range. The model should include the whole range of operation of the choke, such that the ML model can be used for something useful through the whole operation area. But on the other hand, the decision to not include this operation area gave a much better ML model, which can be used to something useful. While a ML model with the whole range of the choke would predict worse than a ML model not including this operation area.

A ML model with more features correlated to the oil rate would give the ML model better performance. As more features would make the ML model found more relationships to the oil rate. The added features can be the pressure measurement BHP and WHP. They are influenced as a direct consequence of the oil rate, but unfortunately they are also affected by the choke. Therefore making a prediction of the oil rate based upon choke, BHP and WHP would be hard as the BHP and WHP isn’t fixed and are changing as the choke changes.

The ML model idea were first designed as an idea to run a ML algorithm besides of the SEPTIC application. It would be trained and predict steady-state gain while SEPTIC were running. As the model is trained now, it’s not trained on live running values from SEPTIC MPC controller, but on step responses. Because of this, a evaluation on the ML model with values from the MPC controller in SEPTIC would be good to do. The

might be a good idea to train the ML model on both data from step responses and data from control of the MPC in SEPTIC.

The implementation of the gain scheduling in SEPTIC were not implemented as first wanted. The wanted idea was instead of having a gain function to set the gain of the model, a OPC-tag sets the gain using predictions from a running ML model in python. But this idea fell apart, since the OPC client object in antiSEPTIC demanded 32-bit python, while tensorflow and the ML model demanded 64 bit python. This means that you can't make predictions on the ML model while in the same time have a OPC client running.

A possible implementation of a ML model with SEPTIC would look something like fig. 28. Here it is a ML server, the server could be the tensorflow's serving functionality. A OPC client which connects to the OPC server and makes request through the TCP/IP communication protocol to the ML server. Such architecture makes sure that communication to/from SEPTIC is done through OPC communication, as this is the prefer communication for Equinor to and from SEPTIC. The benefit of having a ML model running and trained online with SEPTIC is the fact the ML model would get better and better with more run time on SEPTIC. In addition would the ML model detect changes of system dynamics over time, always be up to date on the gain scheduler.

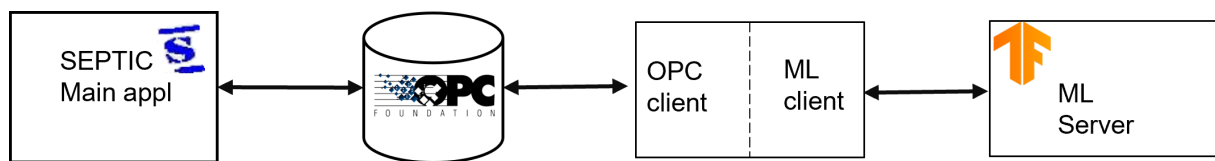


Figure 28: Illustration of the architecture of a ML model implementation

6 Conclusion

The goal of the master thesis was to look into how system dynamics could be developed with TDD. In section 3 a complex sequence were implemented with TDD including testing of some system dynamics. The testing of the system dynamics were successfully, it is important that the developer knows how the simulation plays out and have completely control of where in the simulation one tests the system dynamics. The function *step_until* gives completely control of simulation while testing. All in total it is concluded with it is possible to develop system dynamics with TDD in SEPTIC.

The ML model developed in section 4 successfully modelled the steady-state behaviour between the choke opening and the oil rate. The model generalized very good around 5% step lengths, but the model lacks modeling of choke opening between 0% – 15% but in contrast it models very well in the rest of the operating area. A model with modeling the whole operation area would be preferable, and could maybe achieved by better ML model, this can be seen as further work for later. A gain scheduling is implemented, using the ML model as basis for calculating a gain function. A better implementation, were a running along with SEPTIC would be preferable, as the machine learning model would be trained and be better as more run time of the well progress. In addition would the machine learning model detect gain differences through the years as the oil well system dynamics may change over years. A possible implementation of a running ML model is proposed in fig. 28.

References

- [1] Thomas Solli Koløen. *Test driven development of industrial MPC application*. Tech. rep. NTNU, Dec. 2021.
- [2] Equinor. *Energy*. URL: <https://www.equinor.com/energy>. (downloaded: 02.05.2022).
- [3] Stig Strand and Jan Sagli. “MPC in Statoil-advantages with in-house technology.” In: *International Symposium on Advanced Control of Chemical Processes (ADCHEM)* 37 (Jan. 2003).
- [4] Bjarne Foss and Tor Aksel N Heirung. “Merging optimization and control.” In: *Lecture Notes* (2013).
- [5] HUGO RYVIK. *Supermodell mot fremtiden*. URL: <https://www.tu.no/artikler/supermodell-mot-fremtiden/268926>. (downloaded: 13.05.2022).
- [6] Siddharta Govindaraj. *Test-Driven Python Development*. Community Experience Distilled. Packt Publishing, 2015. URL: <https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=986718&site=ehost-live>.
- [7] pytest. *pytest: helps you write better programs*. URL: <https://docs.pytest.org/en/6.2.x/>. (downloaded: 08.12.2021).
- [8] pytest. *pytest fixtures: explicit, modular, scalable*. URL: <https://docs.pytest.org/en/6.2.x/fixture.html>. (downloaded: 08.12.2021).
- [9] Wolfram. *Functional Mock-up Interface (FMI)*. URL: <https://reference.wolfram.com/system-modeler/UserGuide/ModelCenterFunctionalMockupInterface.html#192378648>. (downloaded: 07.12.2021).
- [10] OPC Foundation. *What is OPC*. URL: <https://opcfoundation.org/about/what-is-opc/>. (downloaded: 15.11.2021).
- [11] OPC Foundation. *Unified Architecture*. URL: <https://opcfoundation.org/about/opc-technologies/opc-ua/>. (downloaded: 15.11.2021).
- [12] Kristin Seter. *Bernoulli-effekten*. URL: <https://snl.no/Bernoulli-effekten>. (downloaded: 25.05.2022).
- [13] Tom M. Mitchell. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997. URL: <https://www.worldcat.org/oclc/61321007>.
- [14] Jason Brownlee. *Overfitting and Underfitting With Machine Learning Algorithms*. URL: <https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/>. (downloaded: 28.05.2022).
- [15] Jason Brownlee. *How to avoid overfitting in DeepLearning Neural Networks*. URL: <https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/>. (downloaded: 31.05.2022).

- [16] Antonio Gulli. *Deep Learning with Keras*. Packt Publishing, 2017. URL: <https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=1510480&site=ehost-live&scope=site>.
- [17] Simplilearn. *What is Perceptron: A Beginners Guide for Perceptron*. URL: <https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron>. (downloaded: 23.05.2022).
- [18] Goodfellow Ian, Bengio Yoshua, and Courville Aaron. *Deep Learning*. Adaptive Computation and Machine Learning. The MIT Press, 2016. URL: <https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=2565107&site=ehost-live&scope=site>.
- [19] Kostadinov Simeon. *Recurrent Neural Networks with Python Quick Start Guide : Sequential Learning and Language Modeling with TensorFlow*. Packt Publishing, 2018. URL: <https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=1950552&site=ehost-live&scope=site>.
- [20] K.E. ArunKumar et al. “Forecasting of COVID-19 using deep layer Recurrent Neural Networks (RNNs) with Gated Recurrent Units (GRUs) and Long Short-Term Memory (LSTM) cells.” In: *Chaos, Solitons & Fractals* 146 (2021), p. 110861. URL: <https://www.sciencedirect.com/science/article/pii/S0960077921002149>.
- [21] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory.” In: *Neural Computation* 9.8 (Nov. 1997), pp. 1735–1780. eprint: <https://direct.mit.edu/neco/article-pdf/9/8/1735/813796/neco.1997.9.8.1735.pdf>. URL: <https://doi.org/10.1162/neco.1997.9.8.1735>.
- [22] Serdar Yegulalp. *What is TensorFlow? The machine learning library explained*. URL: <https://www.infoworld.com/article/3278008/what-is-tensorflow-the-machine-learning-library-explained.html>. (downloaded: 20.05.2022).
- [23] TensorFlow. *Effective Tensorflow 2*. URL: https://www.tensorflow.org/guide/effective_tf2. (downloaded: 21.05.2022).
- [24] Keras. *Keras API References*. URL: <https://keras.io/api/>. (downloaded: 1.06.2022).

A Appendix A: Python code

A.1 step_until.py

Listing 17: step_until.py

```
class stepUntil:

    def __init__(self, director, rui):
        self.rui=rui
        self.director=director

    def step_until(self, appl, xvr, criteria, comparator, value="Meas",
        maxstep=1000):

        N_step=maxstep

        for i in range(maxstep):
            if (comparator(getattr(self.rui.appl[appl][xvr], value), criteria)):
                N_step=i
                break
            else:
                self.director.step(1)

        return N_step
```

A.2 conftest.py

Listing 18: conftest.py

```
import time

import pytest

from antiseptic.opc import OPCClient, OPCServer
from antiseptic.septic import SEPTICClient, SEPTICMaster
from antiseptic.director import DirectorSingle
from antiseptic.septicbase.septicobject import SepticObject
from functions import *

@pytest.fixture(scope="module")
def opc():
    s = OPCServer(
        cfgfile=r"taglist.txt",
        opcserver=r"c:\appl\opcserver\bin\statoilopcserver.exe",
    )
```

```

s.start()
time.sleep(1)
c = OPCClient(scheduletag="Time", pulstag="Heartbeat", steplen=10)
c.connect("Statoil.OPC.Server")
yield c
#c.close()
#s.stop() # Kill the OPC server when test finishes

@pytest.fixture(scope="module")
def main(opc):
    s = SEPTICMaster(
        qtseptic=r"c:\appl\septic\bin\QtSepticFMU.exe",
        cnfgfile=r"C:\appl\septic\Setup\Single_Well_config\SingleWell.cnfg",
        rundir=r"rundir",
    )
    s.start()
    yield s
    #s.stop() # Kill the SEPTIC application when test finishes

@pytest.fixture(scope="module")
def rui(main):
    if not main.reused:
        time.sleep(20)
    c = SEPTICClient()
    c.connect("127.0.0.1", 12030)
    time.sleep(1)
    c.read() # Read INIT msg and initialize rui.appl
    yield c

@pytest.fixture(scope="module", autouse=True)
def director(opc,rui):
    d = DirectorSingle(opc, main=(rui,
        r"C:\appl\septic\Setup\Single_Well_config\SingleWell.cnfg"))
    d.bootstrap(minutes=6)
    opc["Allowactive"]=0
    d.step()
    yield d

@pytest.fixture(scope="module", autouse=True)
def director2(rui,director):
    s=step(director=director,rui=rui)
    yield s

```

A.3 test_sequence.py

Listing 19: test_sequence.py

```

import pytest
from functions import stepUntil
from matplotlib import pyplot as plt
import numpy as np
import operator
from operator import itemgetter

WellOff=-1
WellClosed=1
WellKickOff=2
WellRampUp=3
WellTarget=4
WellShutdown=5

def test_WellOff(opc, rui, director):
    # set mpc to "off" mode
    opc["Allowactive"]=0
    # set Choke to "off" mode
    rui.appl["SingleWell"]["Choke"].Mode=0
    # step to update values in SEPTIC
    director.step(1)
    # Check that the wellstate is in "welloff" state
    assert rui.appl["StartupCalc"]["WellState"].Meas==WellOff

def test_WellClosed(opc, rui, director, director2):

    # Set inital value to choke and gasliftrate
    rui.appl["SingleWell"]["Choke"].Deas=20
    rui.appl["SingleWell"]["GasLiftRate"].Deas=10000

    #obtain stable flow
    director.step(80)

    # check that the well is open
    assert rui.appl["SingleWell"]["OilRate"].Meas>0
    assert rui.appl["SingleWell"]["Choke"].Meas>0

    #Close the well down
    for x in range(21):
        rui.appl["SingleWell"]["Choke"].Deas=20-x #Shut the well down
        director.step(25)

    # step sufficient amount of time such that the system is stable
    director.step(125)
    # check that well is closed and the wellstate is still in WellOff

```

```

assert rui.appl["SingleWell"]["OilRate"].Meas<0.1
assert rui.appl["SingleWell"]["Choke"].Meas==0
assert rui.appl["StartUpCalc"]["WellState"].Meas==WellOff
# Set MPC and choke to active and setpoint to 0
opc["Allowactive"]=1
rui.appl["SingleWell"]["Choke"].Mode=3
rui.appl["SingleWell"]["OilRate"].SetPnt=0
director.step(100)
# Check wellstate is in closed state
assert rui.appl["StartUpCalc"]["WellState"].Meas==WellClosed

def test_WellKickOff(opc, rui, director, director2):
    # Specify minimum choke opening and rate of change of Choke opening
    Zmin=10
    deltaZmaxUp=0.1

    # Set OilRate setpoint to 290
    rui.appl["SingleWell"]["OilRate"].SetPnt=290
    # Step until Wellstate equals WellKickOffState
    N_step=director2.step_until("StartUpCalc", "WellState", WellKickOff,
        comparator=operator.eq)
    #Check that Wellstate is indeed in WellKickOff state
    assert rui.appl["StartUpCalc"]["WellState"].Meas==WellKickOff

    # Step until the choke reaches it's minimum choke value of Zmin
    N_step=director2.step_until("SingleWell", "Choke", Zmin,
        comparator=operator.ge)
    # Check if the choke has reached it's minimum choke value of Zmin
    assert rui.appl["SingleWell"]["Choke"].Meas>=Zmin

    # extract the simulated choke values from septic
    Z=list(map(itemgetter(0), rui.appl["SingleWell"]["Choke"].Meas[-N_step:]))
    # compute the rate of change of the choke numerically
    deltaZ=np.gradient(Z)
    #Check that the rate of change isn't above maximum allowed rate of change
    assert round(max(deltaZ),2)<=deltaZmaxUp

def test_WellRampUp(opc, rui, director, director2):
    # Specify minimum and maximum choke opening, maximum rate of change of
    Choke opening

```

```

# and maximum allowed rate of change in OilRate
Zmin=10
Zmax=100
deltaZmaxUp=0.1
deltaQoilMaxUp=7

#Step until Wellstate reaches WellRampUp state
N_step= director2.step_until("StartUpCalc", "WellState", WellRampUp,
    comparator=operator.eq)
#Check that Wellstate is indeed in WellRampUp state
assert rui.appl["StartUpCalc"]["WellState"].Meas==WellRampUp

#simulate the whole ramp up state to check conditions
N_step= director2.step_until("StartUpCalc", "WellState", WellTarget,
    comparator=operator.eq)

# extract simulated choke and oilrate values from septic
Z=list(map(itemgetter(0), rui.appl["SingleWell"]["Choke"].Meas[-N_step:]))
Qoil=list(map(itemgetter(0),
    rui.appl["SingleWell"]["OilRate"].Meas[-N_step:]))

# compute the rate of change numerically
deltaZ=np.gradient(Z)
deltaQoil=np.gradient(Qoil)

#Check the rate of change isn't above maximum allowed rate of change
assert round(max(deltaQoil),2)<=deltaQoilMaxUp
assert round(max(deltaZ),2)<=deltaZmaxUp

# check if choke is not above or under the boundaries of the choke
assert ((round(min(Z),2)>=Zmin)and(round(max(Z),2)<=Zmax))

def test_WellTargetControl(opc, rui, director, director2):
    # Specify minimum and maximum choke opening
    Zmin=10
    Zmax=100
    deadband=0.5

    #Check that Wellstate is indeed in WellRampUp state
    assert rui.appl["StartUpCalc"]["WellState"].Meas==WellTarget
    # step in WellTarget control to get some values in welltarget
    director.step(200)

    # extract choke and oilrate values from simulation in welltarger
    Z=list(map(itemgetter(0), rui.appl["SingleWell"]["Choke"].Meas[-200:]))
    Qoil=list(map(itemgetter(0), rui.appl["SingleWell"]["OilRate"].Meas[-200:]))

```

```

# check if choke is not above or under the boundaries of the choke
assert ((round(min(Z),2)>=Zmin)and(round(max(Z),2)<=Zmax))
# check if oilrate is not above or under the deadband conition
for x in Qoil: assert ((round(x,2)>290-deadband) and
    (round(x,2)<290+deadband))

# Check transition of states from targetcontrol to rampup and back
rui.appl["SingleWell"]["OilRate"].SetPnt=305
N_step= director2.step_until("StartupCalc", "WellState", WellRampUp,
    comparator=operator.eq)
assert rui.appl["StartupCalc"]["WellState"].Meas==WellRampUp

N_step= director2.step_until("StartupCalc", "WellState", WellTarget,
    comparator=operator.eq)
assert rui.appl["StartupCalc"]["WellState"].Meas==WellTarget
director.step(200)

def test_Shutdown(opc, rui, director, director2):
# Specify maximum down rate of change of Choke opening
# and maximum allowed down rate of change in OilRate
deltaZmaxDn=0.1
deltaQoilDn=7

# set Shutdown condition
rui.appl["SingleWell"]["OilRate"].SetPnt=0.5
# step until wellshutdown state is reached
N_step=director2.step_until("StartupCalc", "WellState", WellShutdown,
    comparator=operator.eq)
# check if the wellstate is indeed wellshutdown
assert rui.appl["StartupCalc"]["WellState"].Meas==WellShutdown

# Step to WellClosed state, such it can test the system dynamics condition
# in well shutdown state
N_step= director2.step_until("StartupCalc", "WellState", WellClosed,
    comparator=operator.eq)

#extract choke opening and oilrate from SEPTIC
Z=list(map(itemgetter(0), rui.appl["SingleWell"]["Choke"].Meas[-N_step:]))
Qoil=list(map(itemgetter(0),
    rui.appl["SingleWell"]["OilRate"].Meas[-N_step:]))

# compute the rate of change numerically
deltaZ=np.gradient(Z)

```

```

deltaQoil=np.gradient(Qoil)

# Check rate of change conditions
assert -round(max(deltaQoil),2)<=deltaQoilDn
assert -round(max(deltaZ),2)<=deltaZmaxDn

# check if the well is indeed closed
assert rui.appl["StartupCalc"]["WellState"].Meas==WellClosed
director.step(100)

```

A.4 test_testSet.py

Listing 20: test_testSet.py

```

def test_testSet(opc, director, rui):
    choke=15
    time=300
    rui.appl["SingleWell"]["Choke"].Deas=choke
    #obtain stable flow
    director.step(500)

    # initialize the process by step up and down from 0% to 100% and back to 15%
    for choke in range(15,105,5):
        rui.appl["SingleWell"]["Choke"].Deas=choke
        director.step(time)

    for choke in range(100,10,-5):
        rui.appl["SingleWell"]["Choke"].Deas=choke
        director.step(time)

    # generate test simulation
    director.step(time)
    rui.appl["SingleWell"]["Choke"].Deas=20
    director.step(time)
    rui.appl["SingleWell"]["Choke"].Deas=24
    director.step(time)
    rui.appl["SingleWell"]["Choke"].Deas=28
    director.step(time)
    rui.appl["SingleWell"]["Choke"].Deas=24
    director.step(time)
    rui.appl["SingleWell"]["Choke"].Deas=27
    director.step(time)
    rui.appl["SingleWell"]["Choke"].Deas=32
    director.step(time)
    rui.appl["SingleWell"]["Choke"].Deas=37
    director.step(time)
    rui.appl["SingleWell"]["Choke"].Deas=33

```



```

director.step(time)
rui.appl["SingleWell"]["Choke"].Deas=37
director.step(time)
rui.appl["SingleWell"]["Choke"].Deas=41
director.step(time)
rui.appl["SingleWell"]["Choke"].Deas=47
director.step(time)
rui.appl["SingleWell"]["Choke"].Deas=40
director.step(time)
rui.appl["SingleWell"]["Choke"].Deas=45
director.step(time)
rui.appl["SingleWell"]["Choke"].Deas=50
director.step(time)
rui.appl["SingleWell"]["Choke"].Deas=56
director.step(time)
rui.appl["SingleWell"]["Choke"].Deas=52
director.step(time)

```

A.5 Convert_Data.py

Listing 21: Convert_Data.py

```

from ReadWriteFile import read_from_file, write_to_file

RawData=[]
RawData=read_from_file(file="MySystem_1.txt")
for x in range(3): # delete the three first lines
    del RawData[0]
RawData.remove("\n")
attributes="," # initial data to be stored from file
data=[]

Skip=0 # sample to skip
for count,x in enumerate(RawData):
    if "Col_" in x :
        word=x.split()
        if (word[1]=="N.sample"):
            continue
        else:
            attributes+=word[1]+","
    elif count<(Skip+len(attributes.split(","))-2):
        continue
    else:
        line=""
        splitData=x.split()
        for i in splitData:
            line+=i+","

```

```

        line=line[:-1]
        data.append(line)

attributes=attributes[:-1]
data.insert(0, attributes)
write_to_file(data,False,"data_set.csv")

```

A.6 ReadWriteFile.py

Listing 22: ReadWriteFile.py

```

import operator

def write_to_file(string, append=False, file="log.txt"):
    if append: mode="a+"
    else: mode="w+"

    f= open(file,mode)
    if isinstance(string, list):
        for x in string:
            f.write(x+"\n")
    else : f.write(string+"\n")
    f.close()

def read_from_file(file="tests/MySystem_1.txt"):
    #content=[]
    f = open(file,'r')
    content=f.readlines()
    f.close()
    return content

```

B Appendix B: SEPTIC configuration files

B.1 WellKickOff calc

Listing 23: Well Kick Off calc

```

DmmyAppl:    StartUpCalc
            Text1= "Calc to start up well"
            Text2= ""
            Nstep= 1
            PlotMax= 25
            MasterOn= ON
            DesMode= ACTIVE

```

```

Evr: WellState Meas= 0

Evr: TransWellClosed Meas= 0
Evr: WellOffState Meas= -1

Evr: WellClosedState Meas= 1
Evr: IsWellClosed
Evr: TransWellOff Meas= 0

Evr: TransKickOff Meas= 0
Evr: WellKickOffState Meas= 2
Evr: IsWellKickOff

Evr: ChokeSetLow
Evr: ChokeSetHigh
Evr: ChokeLow
Evr: ChokeHigh

Evr: ChokeLowKickOff Meas= 10
Evr: ChokeHighKickOff Meas= 11

CalcMod1:    Calc
             Text1= ""
             Text2= ""

//----- Transitions-----
CalcPvr:     TransWellOff
             Text1= ""
             Text2= ""
             Alg= "not(getfinalstatus(Choke)==3)"

CalcPvr:     TransWellClosed
             Text1= ""
             Text2= ""
             Alg= "and(OilRate<0.5,Choke<0.5)"

CalcPvr:     TransKickOff
             Text1= ""
             Text2= ""
             Alg= "and(OilRate.SetPnt>0.5,OilRate<0.5)"

// -----Calculation of WellState-----

CalcPvr:     IsWellClosed
             Text1= ""
             Text2= ""

```

```

    Alg= "or(and(TransWellClosed, not(TransWellOff), not(TransKickOff)),
    and(WellState==WellClosedState,not(TransWellOff),
    not(TransKickOff)))"

CalcPvr:    IsWellKickOff
    Text1= ""
    Text2= ""
    Alg= "or(and(TransKickOff not(TransWellOff)),
    and(WellState==WellKickOffState, not(TransWellOff)))"

CalcPvr:    WellState
    Text1= ""
    Text2= ""
    Alg= "TransWellOff*WellOffState+IsWellClosed*WellClosedState+
    IsWellKickOff*WellKickOffState"

//-----Calculation of Choke high and low
CalcPvr:    ChokeLow
    Text1= ""
    Text2= ""
    Alg= "if(WellState==WellKickOffState,ChokeLowKickOff,0)"

CalcPvr:    ChokeHigh
    Text1= ""
    Text2= ""
    Alg= "if(WellState==WellKickOffState,ChokeHighKickOff,100)"

//-----Set low and high on Choke
CalcPvr:    ChokeSetLow
    Text1= ""
    Text2= ""
    Alg= "setlow(Choke,ChokeLow)"

CalcPvr:    ChokeSetHigh
    Text1= ""
    Text2= ""
    Alg= "sethigh(Choke,ChokeHigh)"

```

B.2 Well Sequence calc

Listing 24: Well Sequence calc

```

DmmyAppl:    StartUpCalc
    Text1= "Calc to start up well"
    Text2= ""
    Nstep= 1

```

```
PlotMax= 25
MasterOn= ON
DesMode= ACTIVE

Evr: TransWellClosed Meas= 0
Evr: TransKickOff Meas= 0
Evr: TransRampUp Meas= 0
Evr: TransTargetControl Meas= 0
Evr: TransShutdown Meas= 0
Evr: TransWellOff Meas= 0

Evr: WellState Meas= 0
Evr: WellOffState Meas= -1
Evr: WellClosedState Meas= 1
Evr: WellKickOffState Meas= 2
Evr: WellRampUpState Meas= 3
Evr: WellTargetControlState Meas= 4
Evr: WellShutdownState Meas= 5

Evr: IsWellKickOff
Evr: IsWellRampUp
Evr: IsWellClosed
Evr: IsWellTargetControl
Evr: IsWellShutdown

Evr: ChokeSetLow
Evr: ChokeSetHigh

Evr: ChokeLow Meas= 0
Evr: ChokeHigh Meas= 100

Evr: ChokeLowKickOff Meas= 10
Evr: ChokeHighKickOff Meas= 11

Evr: ChokeLowWellClosed Meas= 0
Evr: ChokeHighWellClosed Meas= 100

Evr: ChokeLowWellShutdown Meas= 0
Evr: ChokeHighWellShutdown Meas= 100

Evr: ChokeLowWellRampUp Meas= 10
Evr: ChokeHighWellRampUp Meas= 100

Evr: ChokeLowWellTargetControl Meas= 10
Evr: ChokeHighWellTargetControl Meas= 100

Evr: QoilMin Meas= 180
```

```

Evr: Deadband Meas= 0.5
Evr: OilPred

CalcMod1:    Calc
  Text1= ""
  Text2= ""

//-----Transitions-----
CalcPvr:      TransWellOff
  Text1= ""
  Text2= ""
  Alg= "not(getfinalstatus(Choke)==3)"

CalcPvr:      TransShutdown
  Text1= ""
  Text2= ""
  Alg= "and(OilRate.SetPnt<=0.5, OilRate>=0.5, Choke>=0.5)"

CalcPvr:      TransWellClosed
  Text1= ""
  Text2= ""
  Alg= "and(OilRate<0.5,Choke<0.5)"

CalcPvr:      TransKickOff
  Text1= ""
  Text2= ""
  Alg= "and(OilRate.SetPnt>0.5,OilRate<0.5)"

CalcPvr:      TransRampUp
  Text1= ""
  Text2= ""
  Alg= "and((Choke>=ChokeLowKickOff), OilRate>=QoilMin)"
  //
CalcPvr:      TransTargetControl
  Text1= ""
  Text2= ""
  Alg= "and((abs(OilRate-getssval(OilRate)))<Deadband,
        TransRampUp)"

CalcPvr:      Tull
  Text1= ""
  Text2= ""
  Alg= "(abs(OilRate-getssval(OilRate)))"

//-----Calculation of WellState-----

CalcPvr:      IsWellClosed
  Text1= ""
  Text2= ""

```

```

        Alg= "or(and(TransWellClosed, not(TransKickOff),
                not(TransWellOff)),
                and(WellState==WellClosedState,not(TransKickOff),not(TransWellOff)))"

CalcPvr:      IsWellShutdown
Text1= ""
Text2= ""
Alg= "or(and(TransShutdown, not(TransWellOff),
            not(TransWellClosed)),and(WellState==WellShutdownState,not(TransWellOff),
            not(TransWellClosed)))"

CalcPvr:      IsWellKickOff
Text1= ""
Text2= ""
Alg= "or(and(TransKickOff, not(TransRampUp),
            not(TransWellOff), not(IsWellShutdown)),
            and(WellState==WellKickOffState, not(TransRampUp),
            not(TransWellOff), not(IsWellShutdown)))"

CalcPvr:      IsWellRampUp
Text1= ""
Text2= ""
Alg= "or(and(TransRampUp,not(TransTargetControl),
            not(TransWellOff), not(IsWellShutdown)),
            and(WellState==WellRampUpState, not(TransTargetControl),
            not(TransWellOff), not(IsWellShutdown)))"

CalcPvr:      IsWellTargetControl
Text1= ""
Text2= ""
Alg= "or(and(TransTargetControl,not(TransShutdown),
            not(TransWellOff), not(IsWellShutdown)),
            and(WellState==WellTargetControlState,
            not(TransShutdown), not(TransWellOff),
            not(IsWellShutdown)))"

CalcPvr:      WellState
Text1= ""
Text2= ""
Alg= "
        TransWellOff*WellOffState+IsWellClosed*WellClosedState+
        IsWellKickOff*WellKickOffState+IsWellRampUp*WellRampUpState+
        IsWellTargetControl*WellTargetControlState+
        IsWellShutdown*WellShutdownState"

//-----Calculation of Choke high and low
CalcPvr:      ChokeLow

```

```

Text1= ""
Text2= ""
Alg= "if(WellState==WellClosedState,ChokeLowWellClosed,0)+
      if(WellState==WellKickOffState, ChokeLowKickOff,0)+
      if(WellState==WellRampUpState, ChokeLowWellRampUp,0)+i
      f(WellState==WellTargetControlState,
      ChokeLowWellTargetControl,0)+
      if(WellState==WellShutdownState,
      ChokeLowWellShutdown,0)+if(WellState>5,10,0)"

CalcPvr:      ChokeHigh
Text1= ""
Text2= ""
Alg= "if(WellState==WellClosedState,
      ChokeHighWellClosed,0)+if(WellState==WellKickOffState,
      ChokeHighKickOff,0)+if(WellState==WellRampUpState,
      ChokeHighWellRampUp,0)+if(WellState==WellTargetControlState,
      ChokeHighWellTargetControl,0)+if(WellState==WellShutdownState,
      ChokeHighWellShutdown,0)+if(WellState>5,100,0)"

//-----Set low and high on Choke
CalcPvr:      ChokeSetLow
Text1= ""
Text2= ""
Alg= "setlow(Choke,ChokeLow)"

CalcPvr:      ChokeSetHigh
Text1= ""
Text2= ""
Alg= "sethigh(Choke,ChokeHigh)"

```

B.3 SinglWell.cnfg

Listing 25: SEPTIC configuration file

```

System:      MySystem
Text1= "SEPTIC test application"
Text2= ""
Nsecs= 10
PlotMax= 10
Nhist= 25920
Npred= 360
Nmodl= 180
Nxupd= 1
ClipOn= OFF
ChngLogOn= OFF

```



```
WdwDx= 800
WdwDy= 400
Xline= 0
Xback= 0.7
Grps= 5
    "Process1" "Calctable" "Free" "Free" "Free"
GrpLock= 0000
Xgroup= 0001
MaxLogon= ROOT
RootPwd= ""
MngrPwd= ""
OperPwd= ""
UserPwd= ""
Terr= 0
DebugNR= 25920
DebugOn= 11111111
Zname= 225
Zhilo= 275
Zstate= 225
Zopen= 225
Zfont= 275
Zline= 275
FontSize= -1
InitialGroup= -1
ShowText1WithName= ON
MinXPos= 0
MinYPos= 0
MaxXPos= 0
MaxYPos= 0
InitScreen= -1
InitXPos= -1
InitYPos= -1
InitWidth= -1
InitHeight= -1
MinimizeByHide= OFF
HideCloseButton= OFF
OPCSampleTimeOffset= 0

SopcProc:    SingleWell
    Text1= "TestSopcProc antiSEPTIC"
    Text2= ""
    Site= AIM_AIMGUI
    ServName= "Statoil.OPC.Server"
    ServNode= "SERVNODE"
RealTimeFac= 1
ProcTag= ""
```

```

AllowActiveTag= "AllowActive"
  StatusTag= "NotUsed"
  DesModeTag= "DUMMY_TAG"
  PulsTag= "Heartbeat"
LoopcheckWriteTag= "DUMMY_TAG"
LoopcheckReadTag= "DUMMY_TAG"
  ServFactorTag= "DUMMY_TAG"
  CPUTimeTag= "DUMMY_TAG"
  IdTag= ""
  ScheduleTag= "Time"
  RunSec= -1
  RequestRate= -1
  BadCountLimit= 0
  SampleAgeLimit= -1
  BlockWritingIfBad= OFF
  WriteGroups= 1
  WriteGroupPause= 10

```

```

  SopcTvr:      Apply
    Text1= ""
    Text2= ""
    TvrTag= ""
    MeasTag= "Apply"
    NotValidTag= "NotUsed"
    Text1Tag= ""
    Scale= 1
    Offset= 0

```

```

  FMUProc:      SingleWell_FMU
    Text1= "FMU containing 1 Well"
    Text2= ""
    FMUname= "SingleWell.fmu"

```

```
// FMU for well 1.....
```

```

  SubrXvr:      Choke
    Text1= "Choke position"
    Text2= " "
    DtaIx= "u[1]"
    Init= 0

```

```

  SubrXvr:      GasLiftRate
    Text1= "Gas lift rate"
    Text2= " "

```

```
DtaIx= "u[2]"
Init= 0

SubrXvr:    TopsidePressure
Text1= "Inlet pressure at topside separator"
Text2= " "
DtaIx= "u[3]"
Init= 0

SubrXvr:    GasRate
Text1= ""
Text2= " "
DtaIx= "y[1]"
Init= 0

SubrXvr:    OilRate
Text1= "Oil Rate"
Text2= " "
DtaIx= "y[2]"
Init= 0

SubrXvr:    WaterRate
Text1= ""
Text2= " "
DtaIx= "y[3]"
Init= 0

SubrXvr:    GasLiftRatePV
Text1= ""
Text2= " "
DtaIx= "y[4]"
Init= 0

SubrXvr:    BHP
Text1= "Bottom hole pressure"
Text2= " "
DtaIx= "y[5]"
Init= 182

SubrXvr:    WHP
Text1= "Well head pressure"
Text2= " "
DtaIx= "y[6]"
Init= 26

SubrXvr:    DCP
Text1= ""
Text2= " "
```

```
DtaIx= "y[7]"
Init= 16

SubrXvr:    PGL
Text1= ""
Text2= " "
DtaIx= "y[8]"
Init= 166

SubrXvr:    ChokePV
Text1= ""
Text2= " "
DtaIx= "y[9]"
Init= 0

SubrXvr:    GOR
Text1= ""
Text2= " "
DtaIx= "y[10]"
Init= 29

SubrXvr:    Velocity
Text1= "DSC velocity"
Text2= " "
DtaIx= "y[11]"
Init= 2.76

MasterTcip: MyApplication
MasterPort= 12030
WriteHosts= 1 "127.0.0.1"

DmmyAppl:   StartUpCalc
Text1= "Calc to start up well"
Text2= ""
Nstep= 1
PlotMax= 25
MasterOn= ON
DesMode= ACTIVE

Evr: TransWellClosed Meas= 0
Evr: TransKickOff Meas= 0
Evr: TransRampUp Meas= 0
Evr: TransTargetControl Meas= 0
Evr: TransShutdown Meas= 0
Evr: TransWellOff Meas= 0

Evr: WellState Meas= 0
```

```

Evr: WellOffState Meas= -1
Evr: WellClosedState Meas= 1
Evr: WellKickOffState Meas= 2
Evr: WellRampUpState Meas= 3
Evr: WellTargetControlState Meas= 4
Evr: WellShutdownState Meas= 5

Evr: IsWellKickOff
Evr: IsWellRampUp
Evr: IsWellClosed
Evr: IsWellTargetControl
Evr: IsWellShutdown

Evr: ChokeSetLow
Evr: ChokeSetHigh

Evr: ChokeLow Meas= 0
Evr: ChokeHigh Meas= 100

Evr: ChokeLowKickOff Meas= 10
Evr: ChokeHighKickOff Meas= 11

Evr: ChokeLowWellClosed Meas= 0
Evr: ChokeHighWellClosed Meas= 100

Evr: ChokeLowWellShutdown Meas= 0
Evr: ChokeHighWellShutdown Meas= 100

Evr: ChokeLowWellRampUp Meas= 10
Evr: ChokeHighWellRampUp Meas= 100

Evr: ChokeLowWellTargetControl Meas= 10
Evr: ChokeHighWellTargetControl Meas= 100

Evr: QoilMin Meas= 180
Evr: Deadband Meas= 0.5
Evr: OilPred

    CalcMod1:    Calc
                Text1= ""
                Text2= ""

    //-----Transitions-----
    CalcPvr:     TransWellOff
                Text1= ""
                Text2= ""
                Alg= "not(getfinalstatus(Choke)==3)"

```

```
CalcPvr:      TransShutdown
  Text1= ""
  Text2= ""
  Alg= "and(OilRate.SetPnt<=0.5, OilRate>=0.5, Choke>=0.5)"

CalcPvr:      TransWellClosed
  Text1= ""
  Text2= ""
  Alg= "and(OilRate<0.5,Choke<0.5) "

CalcPvr:      TransKickOff
  Text1= ""
  Text2= ""
  Alg= "OilRate.SetPnt>0.5"

CalcPvr:      TransRampUp
  Text1= ""
  Text2= ""
  Alg= "and((Choke>=ChokeLowKickOff), OilRate>=QoilMin)"
  //

CalcPvr:      TransTargetControl
  Text1= ""
  Text2= ""
  Alg= "and((abs(OilRate-getssval(OilRate)))<Deadband,
  TransRampUp) "

//-----Calculation of WellState-----

CalcPvr:      IsWellClosed
  Text1= ""
  Text2= ""
  Alg= "or(and(TransWellClosed, not(TransKickOff),
  not(TransWellOff)), and(WellState==WellClosedState,
  not(TransKickOff),not(TransWellOff)))"

CalcPvr:      IsWellShutdown
  Text1= ""
  Text2= ""
  Alg= "or(and(TransShutdown, not(TransWellOff),
  not(TransWellClosed)), and(WellState==WellShutdownState,
  not(TransWellOff), not(TransWellClosed)))"

CalcPvr:      IsWellKickOff
  Text1= ""
  Text2= ""
```

```

    Alg= "or(and(TransKickOff, not(TransRampUp),
              not(TransWellOff), not(IsWellShutdown)),
          and(WellState==WellKickOffState, not(TransRampUp),
              not(TransWellOff), not(IsWellShutdown)))"

CalcPvr:      IsWellRampUp
Text1= ""
Text2= ""
Alg= "or(and(TransRampUp,not(TransTargetControl),
            not(TransWellOff), not(IsWellShutdown)),
        and(WellState==WellRampUpState, not(TransTargetControl),
            not(TransWellOff), not(IsWellShutdown)))"

CalcPvr:      IsWellTargetControl
Text1= ""
Text2= ""
Alg= "or(and(TransTargetControl,not(TransShutdown),
            not(TransWellOff), not(IsWellShutdown)),
        and(WellState==WellTargetControlState,
            not(TransShutdown), not(TransWellOff),
            not(IsWellShutdown)))"

CalcPvr:      WellState
Text1= ""
Text2= ""
Alg= "TransWellOff*WellOffState+IsWellClosed*WellClosedState
      +IsWellKickOff*WellKickOffState+IsWellRampUp*WellRampUpState
      + IsWellTargetControl*WellTargetControlState+
      IsWellShutdown*WellShutdownState"

//-----Calculation of Choke high and low
CalcPvr:      ChokeLow
Text1= ""
Text2= ""
Alg= "if(WellState==WellClosedState,ChokeLowWellClosed,0)+
      if(WellState==WellKickOffState,ChokeLowKickOff,0)+
      if(WellState==WellRampUpState,ChokeLowWellRampUp,0)+
      if(WellState==WellTargetControlState,
        ChokeLowWellTargetControl,0)+
      if(WellState==WellShutdownState,ChokeLowWellShutdown,0)+
      if(WellState>5,10,0)"

CalcPvr:      ChokeHigh
Text1= ""
Text2= ""
Alg= "if(WellState==WellClosedState,ChokeHighWellClosed,0)+
      if(WellState==WellKickOffState,ChokeHighKickOff,0)+
      if(WellState==WellRampUpState,ChokeHighWellRampUp,0)+

```

```

        if(WellState==WellTargetControlState,
        ChokeHighWellTargetControl,0)+
        if(WellState==WellShutdownState,ChokeHighWellShutdown,0)+
        if(WellState>5,100,0)"

//-----Set low and high on Choke
CalcPvr:      ChokeSetLow
        Text1= ""
        Text2= ""
        Alg= "setlow(Choke,ChokeLow)"

        CalcPvr:      ChokeSetHigh
        Text1= ""
        Text2= ""
        Alg= "sethigh(Choke,ChokeHigh)"

// SMPC for well.....
SmpcAppl:      SingleWell
        Text1= "MIMO MPC for Well 1 of Cvr: Oil Rate, WHP and BHP with Mvr:
        Choke and GasLiftRate"
        Text2= ""
        Npred= 360
        Nstep= 1
        PlotMax= 25
        Nhorz= 360
        Nstart= 0
        MasterOn= ON
        DesMode= TRACKING
        FailMax= 0
        PriceOn= ON
        PriceScale= 1
        IterOpt= OFF
        IterNewSens= OFF
        IterQpMax= 10
        IterLineMax= 0
        DoStdSolve= ON
        SteadySolver= QP
        MajItLim= 200
        MajPrint= 0
        ObjConPrint= OFF
        VerifyGrads= OFF
        FuncPrec= 9.9999999e-009
        FeTol= 9.9999997e-006
        OptimTol= 1e-006
        FdifIntv= 0.001
        MaxSeconds= 10
        LmPrio= 100

```



```

MaxPrioSQP= 100
UnConstrnd= OFF
  OpenFlag= OPTMVR
  UpdFilt= 0
  RelPert= 0.050000001
  FeasTol= 1e-007
  EachParam= OFF
LinErrorLim= 1000
  ColdStart= OFF
SimOptimal= OFF
  PrintSens= 0
  SensLimSS= 1e-006
  SensLimDyn= 1e-006

// S MPC for well1}.....
Cvr:      WHP
  Text1= "Well head pressure"
  Text2= ""
  Mode= TRACKING
  Auto= OFF
  PlotMax= 80
  PlotMin= 10
  PlotSpan= -1
  PlotGrp= 00000000000000000000000000000000
  Nfix= 1
  MaxChg= -1
  Unit= "bar"
  Meas= 19.3
  GrpMask= 000000000000000000000000000000001000000000
  GrpType= 00000000000000000000000000000000000000000000
  Span= 1
  SetPntOn= 0
  HighOff= 25
  LowOff= 17
  SetPntPrio= 2
  HighPrio= 2
  LowPrio= 1
  HighBackOff= 0
  LowBackOff= 0
  Fulf= 1
  HighPnlty= 1
  LowPnlty= 10
  HighLimit= 1000
  LowLimit= 1000
  RelxParam= 3   1   30   80
  FulfReScale= 0.001
  SetpTref= 0

```

```

BiasTfilt= 0
BiasTpred= 0
Constfilt= -1
  Integ= 0
TransformType= NOTRANS
  BadCntLim= 0
  DesHorz= 0
  Neval= 5
  EvalDT= 0
  KeepTargets= OFF
MeasValidation= ON
MeasHighLimit= 100000
MeasLowLimit= -10
  LockHL= OFF
  LockSP= OFF
  LockLL= OFF
UseFactorWeight= 0

Cvr:      OilRate
  Text1= "Oil Rate"
  Text2= ""
  Mode= ACTIVE
  Auto= OFF
  PlotMax= 420
  PlotMin= -5
  PlotSpan= -1
  PlotGrp= 00000000000000000000000000000000
  Nfix= 1
  MaxChg= -1
  Unit= "Sm3/hr"
  Meas= 19.3
  GrpMask= 0000000000000000000000000100000000
  GrpType= 00000000000000000000000000000000
  Span= 1
  SetPntOn= 0
  HighOff= 25
  LowOff= 17
  SetPntPrio= 1
  HighPrio= 2
  LowPrio= 1
  HighBackOff= 0
  LowBackOff= 0
  Fulf= 4
  HighPnlty= 1
  LowPnlty= 10
  HighLimit= 1000
  LowLimit= 1000
  RelxParam= 3  1  30  80

```

```
FulfReScale= 0.001
  SetpTref= 0
  BiasTfilt= 0
  BiasTpred= 0
  ConstTfilt= -1
    Integ= 0
TransformType= NOTRANS
  BadCntLim= 0
  DesHorz= 0
  Neval= 5
  EvalDT= 0
  KeepTargets= OFF
MeasValidation= ON
MeasHighLimit= 100000
MeasLowLimit= -10
  LockHL= OFF
  LockSP= OFF
  LockLL= OFF
UseFactorWeight= 0
  Cvr:      BHP
    Text1= "Bottom hole pressure"
    Text2= ""
    Mode= TRACKING
    Auto= OFF
    PlotMax= 190
    PlotMin= 165
    PlotSpan= -1
    PlotGrp= 00000000000000000000000000000000
      Nfix= 1
      MaxChg= -1
      Unit= "bar"
      Meas= 183.44604
    GrpMask= 000000000000000000000000000000001000000000
    GrpType= 00000000000000000000000000000000000000000000
      Span= 1
    SetPntOn= 0
    HighOff= 185
    LowOff= 155
  SetPntPrio= 2
    HighPrio= 2
    LowPrio= 2
  HighBackOff= 0
  LowBackOff= 0
    Fulf= 1
  HighPnlty= 1
  LowPnlty= 1
  HighLimit= 1000
  LowLimit= 1000
```



```

    Price= 0
    Blocking= 9  1  2  4  8  16  32  64  128  256
MeasValidation= ON
MeasHighLimit= 105
MeasLowLimit= -5
    LockHL= OFF
    LockIV= OFF
    LockLL= OFF
UseFactorWeight= 0

Mvr:          GasLiftRate
    Text1= "Gas lift rate"
    Text2= ""
    Mode= TRACKING
    Auto= OFF
    PlotMax= 15000
    PlotMin= 0
    PlotSpan= -1
    PlotGrp= 00000000000000000000000000000000
    Nfix= 0
    MaxChg= -1
    Unit= "Sm3/hr"
    Meas= 0
    GrpMask= 0000000000000000000000000100000000
    GrpType= 00000000000000000000000000000000
    Span= 100
    HighOn= 15000
    LowOn= 0
ProcessValue= 293.70999
    IvOff= 0
    MaxUp= 1000
    MaxDn= -1000
    MovePnlty= 1
    IvRoc= 1
    IvPrio= 99
    Fulf= 100
    FulfReScale= 0
    Price= 0
    Blocking= 9  1  2  4  8  16  32  64  128  256
MeasValidation= ON
MeasHighLimit= 105
MeasLowLimit= -5
    LockHL= OFF
    LockIV= OFF
    LockLL= OFF
UseFactorWeight= 0

Dvr:          TopsidePressure

```

```
Text1= "Inlet pressure at topside separator"
Text2= ""
Mode= TRACKING
PlotMax= 100
PlotMin= 0
PlotGrp= 00000000000000000000000000000000
XvrMnu= 00000000
Nfix= 1
Unit= "bar"
Meas= 13
GrpMask= 00000000
GrpType= 00000000
Span= 1.0

Tvr:      GasRate
Text1= "Gas Rate"
Text2= ""
PlotMax= 36000
PlotMin= 10000
PlotGrp= 00000000000000000000000000000000
XvrMnu= 00000000
Nfix= 0
Unit= "Sm3/hr"
Meas= 16987
GrpMask= 00000000
GrpType= 00000000
Span= 10

ExprModl:  model1

DisplayGroup: Process
GroupNo= 1
Locked= OFF
Rows= 2
Cols= 3
xGrid= OFF
yGrid= OFF
xAxis= OFF
yAxis= ON
Autoscale= OFF
Spanscale= OFF
HistSize= -1

XvrPlot:      OilRate
Row= 1
```

```
Col= 1

XvrPlot:      WHP
  Row= 1
  Col= 2

XvrPlot:      BHP
  Row= 1
  Col= 3

XvrPlot:      Choke
  Row= 2
  Col= 1

XvrPlot:      GasLiftRate
  Row= 2
  Col= 2

CalcTable:    Calc
  Row= 2
  Col= 3
  RowSize= 1
  ColSize= 1

DmmyAppl:     ModelGainCalc
  Text1= "Calc to update gain in experimental model"
  Text2= ""
  Nstep= 1
  PlotMax= 25
  MasterOn= ON
  DesMode= ACTIVE

Tvr: Apply

Evr: Gain
Evr: GainScheduler
  CalcModl:    ModGain
    Text1= ""
    Text2= ""

  CalcPvr:     Gain
    Text1= ""
    Text2= ""
    Alg= "intpoltype1(Choke,7.5,19.25,17.5,3.46, 22.5,2.134,
          27.5,1.35, 32.5,0.886,37.5,0.603,42.5,
          0.424,47.5,0.306,52.5,0.226,57.5,
```

```
0.170,62.5,0.129,67.5,0.100, 72.5,0.0778,77.5,0.0607,  
82.5,0.0475,87.5,0.0371, 92.5,0.0289,97.5,0.0224)"
```

```
CalcPvr:      GainScheduler  
Text1= ""  
Text2= ""  
Alg= "modgain(OilRate,Choke,Gain,Apply)"
```

C Appendix C: Machine Learning python code

C.1 ML_import.py

Listing 26: ML_import.py

```
# import different packages and functions  
import pandas as pd  
import numpy as np  
import tensorflow as tf  
from tensorflow import keras  
from numpy.random import seed  
from keras.models import Sequential  
from keras.layers import Dense  
from keras.layers import LSTM  
from keras.layers import Dropout  
from keras.callbacks import EarlyStopping  
from keras.models import load_model  
from keras import layers  
from keras import metrics  
from MinMaxScaler import MinMaxScale, MinMaxDeScale
```