FACULTY OF SCIENCE AND TECHNOLOGY

MASTER THESIS

Study programme / specialisation: Computational Engineering

The spring semester, 2022..

Open / Confidential : Open

Author: Maaz Bin Alim

....

(signature author)

Course coordinator: Aksel Hiorth

Supervisor(s): Aojie Hong

Thesis title: Probabilistic decline curve analysis with multiple models

Credits (ECTS): 30

Keywords: Decline curve analysis, Bayesian approach, Akaike information criterion, Model ranking and selection, Arps model, Stretched exponential model, Logistic growth model, Capacitance-resistance model.

Pages: ......42............

+ appendix: ...33........

Stavanger, .15.06/2022...
date/year

Abstract

Numerical models have been established to help understand the longevity of projects when exploring and drilling for hydrocarbons. They aid in understanding and optimizing decisions on the long-term feasibility of a project by using existing data to run simulations and estimate how long a supply can be sustained before its depletion. This thesis aims to explore and evaluate how different models compare to one another, how each model varies against the other, and ultimately decide how to better optimize production forecasts. It also investigates two approaches to the problem namely deterministic and probabilistic.

As petroleum production has increased, there has been a need for companies to meet demands and that requires more accurate methods for reservoir evaluation and forecasting. Numerical models can help in this case but which model to choose and to what extent can they be relied upon are uncertain factors. This thesis aims to answer that question.

The challenge in this thesis is comparing probabilistic and deterministic approaches which has not been done in other research before using real field data.

Contents

Table of Figures

1. Introduction

The production rate of oil and gas typically decreases with time. This is where Decline Curve Analysis (DCA) can be useful. The decline curve analysis has been widely regarded as an industry-accepted method for forecasting hydrocarbon production. One reason for this is it uses data that can easily be obtained and does not require many parameters to produce useful results (Hong et al., 2019). However, choosing which model to proceed with for what situation remains a question without any tangible answer.

Review of previous work

Numerical models to simulate and forecast oil production have been used since as early as the 1910s (Shabib-Asl & Plaksina, 2019). Since then, many models have been derived and adopted some of which include the Arpes model, power-law exponential model, Stretched Exponential Model (SEM), Duong, and the Logistic Growth Model (LGM) The idea of combining forecasts was introduced in the 1970s. and was only until recently where using different models came into consideration for oil production

State of art

Currently, there are different models adopted by companies for different purposes. The model used depends on the scenario and the complexity. The situation as of today is understanding the most suitable model according to production requirements and feasibility. By far the most popular model is the Arpes model. However, it is not reliable in predicting trends for unconventional reservoirs and tends to be more optimistic in these situations (Shabib-Asl & Plaksina, 2019) Therefore, selecting the most suitable model is important. The Arpes method, despite being one of the more popular models, assumes many variables as constants. One such example is in the skin factor or the bottom hole pressure, both of which are taken as constants, and thus in the case of a shale reservoir, it also tends to overestimate the oil production. (Shabib-Asl & Plaksina, 2019).

Most companies to this day depend on a single model on a probabilistic basis. Forecasting often relies on the selection of a single model with the criteria that it is the best fit.

Currently, there are numerous methods to select the best model however, there is no concrete general standard for the selection of a model.

Some of the methods used today are as follows:

- **Appearance**: This method is a subjective approach used by an expert usually on no quantifiable basis for model selection.
- **Maximum likelihood**: model selection based on which model fits the data best.
- **AIC**: A deterministic approach where models are ranked based on their AIC index.
- **Probabilistic**: Making use of weighted probabilities in order to determine the best model.

This thesis will focus on the last two by investigating the rankings of each.

Problem Definition

There is the existing concept that several models can be used for the same scenario. However, stated by Box (1979) "all models are wrong, but some are useful". Naturally one of the questions that arise keeping this in mind is, "Which model is the best?" This question is more subjective than it initially appears because the term "best" is not well-defined. In many applications, the model that best represents the data is regarded as the best model and there is no one model that fits all data (Hong et al., 2019)

As shown in Figure 1, we see an example where 4 different models are tested against the same scenario in which we can see the red marks indicate the likely "truth" value according to the specific model in comparison to the actual value that is at the 50 Mbbl mark.

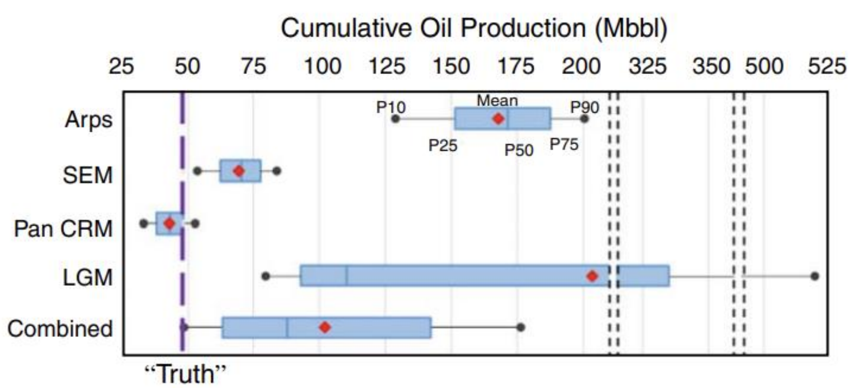


Figure 1-1 Comparison of multiple models against a single truth value (Hong et al., 2019)

Proposed Method

Based on existing decline curve parameters of an SEM model, a randomly generated data set is created. Using this as a reference, the models of SEM, Arpes, LGM, and Pan CRM are implemented accordingly as part of the probabilistic approach. Results obtained from this process are then used to calculate the maximum likelihood of each model, and then are ranked according to their respective probabilities.

As a parallel, we will use the AIC method, as a deterministic approach, to simultaneously compare the two results.

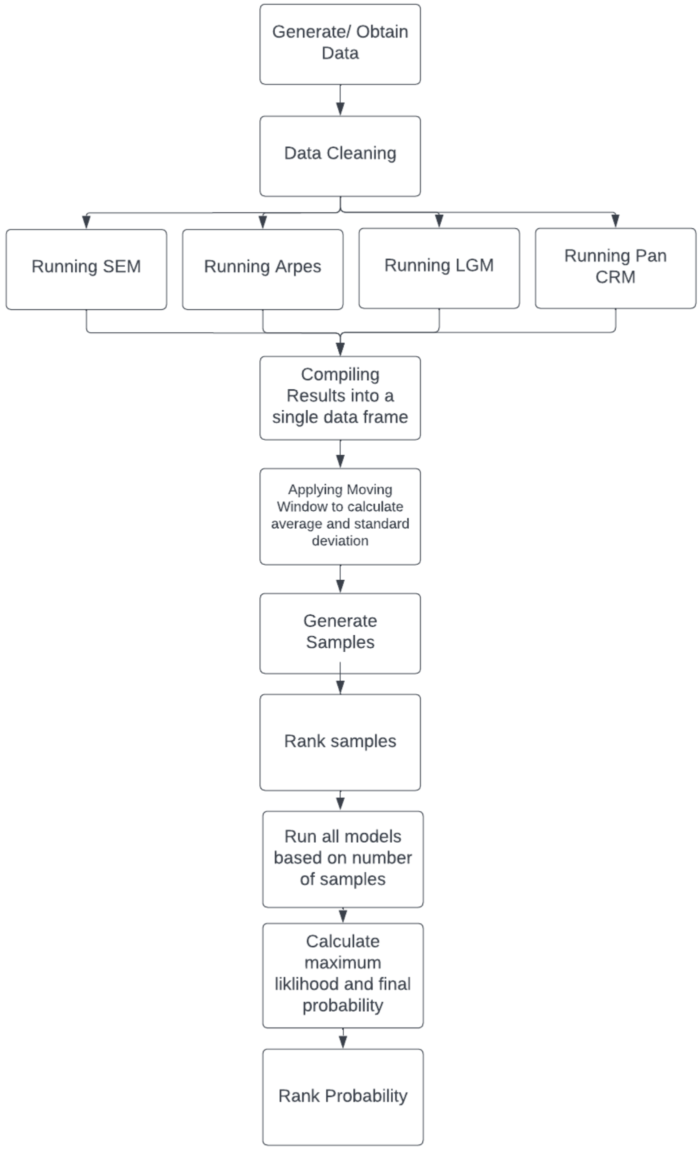


Figure 1-2 Flow chart detailing workflow for the probabilistic approach

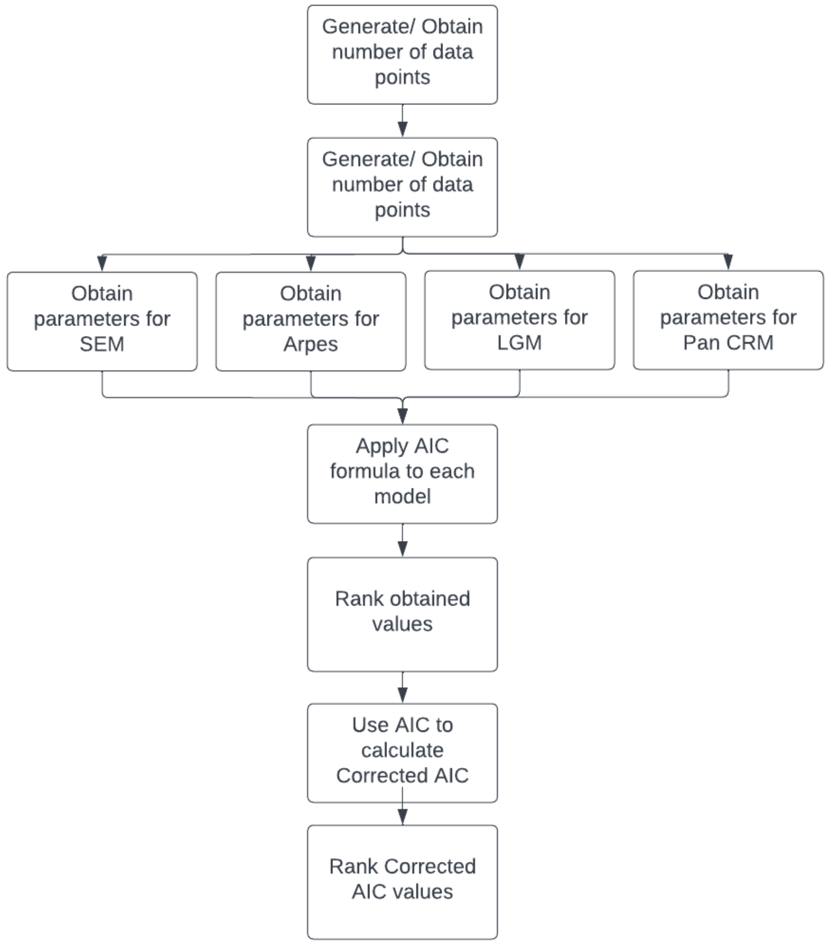


Figure 1-3 Flow chart detailing workflow for the AIC approach

Aim

Objective

Overview of this report

This chapter gives the overview and introduction that defines our motivation, reasons, and eventual objective we are trying to achieve with this thesis.

The second chapter delves deeper into the background of the field, the methodologies and eventual progression and advancements in the field, and as to why we are using these methods and more specifically.

Chapter three will highlight our approach. What tools, formulas, direction, and parameters or restrictions we will work within

Chapter four will detail how our experimentation was conducted and will go through the process on a step-by-step basis on how our experimentation was conducted on synthetic data

Chapter five will investigate our experimental approach to our real data sets, discuss the challenges faced that were not present in the synthetic evaluation

Chapter six will discuss the findings from our work

Chapter seven will investigate the comparison of our results, discuss them, and will provide an eventual recommendation for possible future work.

2. Background

This chapter will investigate the historic progress that has led us to this point.

About Decline Curve Analysis

Decline curves have become a more widely applied and more feasible approach to understanding the potential of production for wells for which not a lot of information may be known. The absence of properties or measurements such as the fluid analysis, bottom hole pressure, and other more advanced techniques and therefore may be more cost-bearing for other approaches therefore decline curves are a more natural choice for forecasting. (Poston & Poe, 2007)

The most widely and extensively used technique used by engineers is the rate-time decline curve. This helps in understanding the forecasting prediction, well performance and fluid prediction. This understanding can contribute significantly towards key economic decisions relating to aspects such as the decision on securing assets, and investment planning.

More sophisticated models can rely on liquid curves in correspondence with pseudo-pressure and pseudo-time alongside empirical curve fitting of the Rate-Time data. (Luis F. & Ye, 2012)

About Probabilistic and Deterministic Approaches

Probabilistic methods include elements and consideration of some form of variation. This means on running the same model multiple times, a different outcome can be expected even with the same set of initial conditions. All probabilistic models incorporate some form of random variation.

On the contrary, deterministic methods are constructed such that elements of randomness are not kept into consideration. This means that if any deterministic model is run any number of times, the result will always be the same.

Statistical statements that do not mention or consider any form of variation can be considered to be deterministic. In this thesis, the AIC and the Corrected AIC method can be viewed as deterministic approaches.

Arpes Model

Based on Johnson and Bollens (1927), The Arpes model aimed to develop a new DCA to estimate reservoirs and generated a relatively simple mathematical formula that, to this day, is widely used especially in the oil and gas industry. However, this model is based on several assumptions such as a constant value of skin factor, bottom-hole pressure, and a uniform BDF regime.

However, Arpes models may tend to overestimate EUR in unconventional wells (Akbarnejad-Nesheli et al., 2012; Okouma Mangha et al., 2012).

The Arpes model bases itself as completely empiric and has three variations, namely: Exponential, Hyperbolic, and Harmonic, each of which can be described as:

Exponential: where b = 0 $$q_0 = q_i \, Dt \exp(-D_i \, t) \tag{1}$$

Hyperbolic: where 0 < b < 1 $$q_0 = \frac{qi}{(1+bD_i t)^{\frac{1}{b}}} \tag{2}$$

Harmonic: where b=1 $$q = = \frac{qi}{(1+bD_i t)} \tag{3}$$

SEM

The SEM production decline model (Valkó, 2009) is regarded as a performance prediction tool set up on an intuitive physical basis. Kohlrausch in 1847 described it as a quantity which is generated by a sum of pure exponential decays with a “fat-tailed” probability distribution. This later was rethought by Valko and Lee (2010) as a determination of many contributing volumes individually.

$$q(t) = q_i \exp\left[-\left(\frac{t}{\tau}\right)^n\right]$$

(4)

In comparison to the Arpes model, SEM has numerous advantages. Some of these advantages include the bounded consideration of EUR from individual wells and the behaviour of recovery potential against the cumulative production (Valko and Lee 2010).

LGM

Originally stated to work as a means to measure and forecast population growth in the Unite States of America, the use of LGM goes back to 1838 as the Verhulst-Pearl Equation (Verhulst, 1838). Until eventually getting repurposed to act as an indicator for production of hydrocarbons. Particularly this model was applied to estimate gas reserves and the model uncertainty for probabilistic DCA was investigated by Hong et al. (2018) for unconventional wells alongside the Arpes model and showed how both, Arpes and LGM may tend to experience more optimistic estimates for production.

The equation of the LGM model can be defined as:

$$qt = \frac{aK\eta^{\eta-1}}{(a + t^{\eta})^2}$$

(5)

Q is the cumulative production. K is defined to be the carrying capacity, η is described as the hyperbolic exponent.

Pan CRM

The Pan Capacitance Resistance Model (Pan CRM) is our final DCA model for the scope of this thesis. Specifically for unconventional wells, it is designed to calculate flow regime for transient and semi-steady state flow regimes. (Pan, 2016) proposed the model to calculate to investigate the productivity index over the different flow regimes which is described by:

$$J = \frac{\beta}{\sqrt{t}} + J_{\infty}$$

6)

J is the productivity, $J\infty$ is the constant of the productivity index that is expected to be reached by the well when there is boundary dominated flow. β represents the linear transient flow and is related to the permeability in the analytical solution as put forward by. The empirical solution was obtained and has assumed the form of the below equation:

$$q_t = \Delta P(\frac{\beta}{\sqrt{t}} + J_\infty)e^{-(2\beta\sqrt{t} + J_\infty t)/c_t V_p} \tag{7}$$

Here the c_t value is the compressibility, V_p represents the drainage pour volume, while ΔPΔP is the difference between the initial reservoir pressure and the constant assumed for the flowing bottom hole pressure.

It is worth noting that for smaller values of time, the Pan CRM model may give unrealistically high values of the production rate as it approaches infinity when time approaches 0.

AIC

The Akaike Information Criterion (AIC) is a deterministic method for calculating the fit of a model against the data it is based on. The method is used to analyse multiple possible models to see which one fits best on the given data. The principle behind AIC can be based on two statements, the first one being the number of independent variables being used in the model we are evaluating. The second being how well the model may reproduce the given data based on the number of data points used.

Using this algorithm, the best model is the one that covers the most variation in the data with the least number of parameters used.

$$AIC = Nln\left(\frac{SS}{N}\right) + 2K \tag{8}$$

N is the number of observations i.e. total datapoints, K is the number of parameters fitted plus one and ss is the sum of squares. According to (Shabib-Asl & Plaksina, 2019), a better approach to calculating the Akaike Criterion would be to use the corrected AIC method.

This method makes use of the calculated AIC value and computes it further as given by the equation:

$$AIC_c = AIC + \frac{2K(K+1)}{N-K-1} \tag{9}$$

The corrected AIC can only be calculated if the number of data points is at least twice as great as the number of parameters being used.

To compare models, the AIC and Corrected AIC values must be calculated for all models being used. The lower the AIC value the better the model fit.

3. Approach

Clemen et al.(2000) concluded that using the average from multiple experts as opposed to just relying on one expert gave a more accurate forecast. This thesis intends to make use of that. The methodology includes using test data and running, simulating, and observing the results of each of the before-mentioned models. Namely:

- SEM
- Arpes
- LGM
- Pan CRM

After running the simulations and collecting results, they are visualized to compare their differences, an example of which, shown in figure 3.1 where the decline curve forecast for each model on the same data set shows little to significant variation.

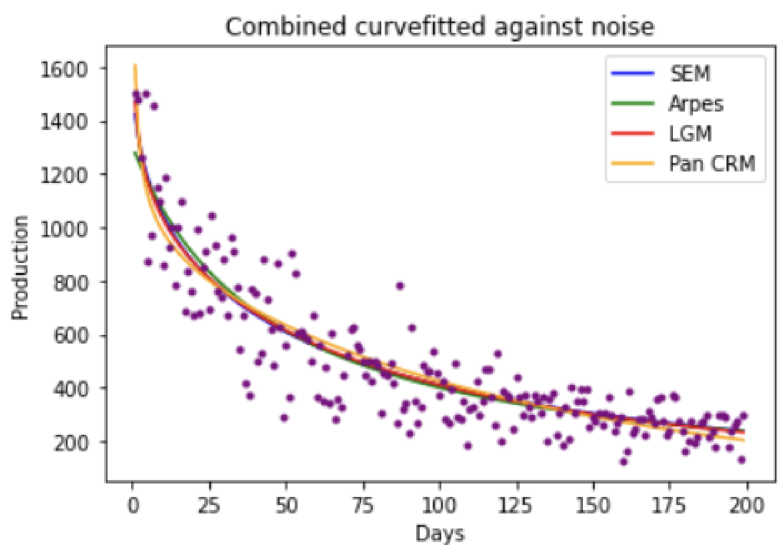


Figure 3-1 Combined representation of all models against original data

Our approach makes use of not only comparing the models based on one data set alone but making use of multiple samples based on the standard deviation obtained through a moving window function. It is expected that the length of data will have to be reduced as well due to the nature of implementing the moving window approach for standard deviation. This may cause some initial time values to be rendered as invalid, or as NaN values in python code.

This approach will also need the data to be as smooth as possible. In real world cases operational changes, testing or accidents may cause some inconsistencies in production data. One of the assumptions used in implementing DCA is that it requires a smooth curve. For this reason, some values in our data pre-processing will be removed from the processed data.

Since the data we are using from real word cases are conducted over different wells with varying decline curves, it is possible that the number of days over production will not be the same for all the wells tested. Some decline curves can span over longer durations than others. It is therefore also important to consider the days needed for production forecasting. The probability of a model may change significantly if the length of time is reduced thus increasing the uncertainty of the model. The data used for our real-world testing can be seen below where out of the 13 wells tested the minimum, maximum, and the average length of time are described:

Reading	Length (Number of data points t)
Minimum	69
Maximum	97
Average	84.53

Table 3.1: Variation of time variable of tested wells

4. Study with synthetic data

The approach taken to finding out the comparison was taken in two phases for both methodologies. For the synthetic data, we first conduct the probabilistic analysis and after obtaining the results and ranking the respective probabilities, we will then conduct the AIC method.

Data required

Using random generation for data points

The first part of this section incorporates generating normally distributed data points over the time of 200 days using existing parameters from an SEM model. Once done with a standard deviation of 0.2, the resulting data trend was as shown in figure 4.1 below:

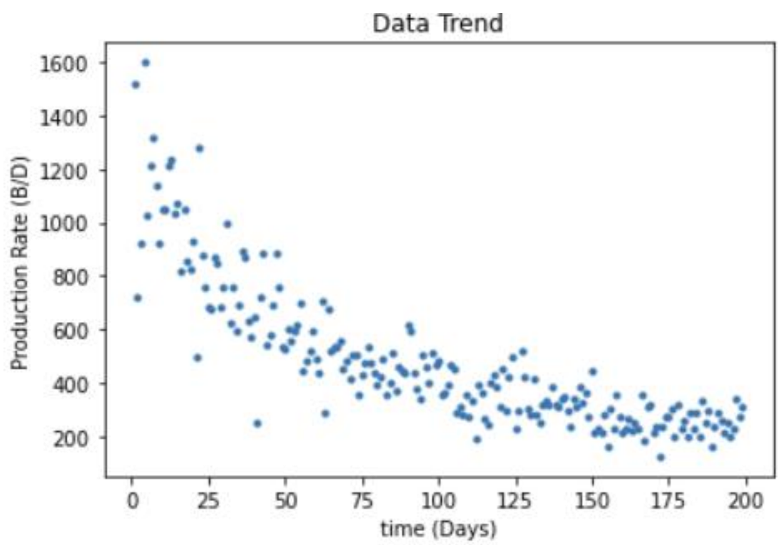


Figure 4-1 Trend of synthetic data

To get a better look of the trend, a line of best fit was also incorporated, as shown in figure 4.2 below:

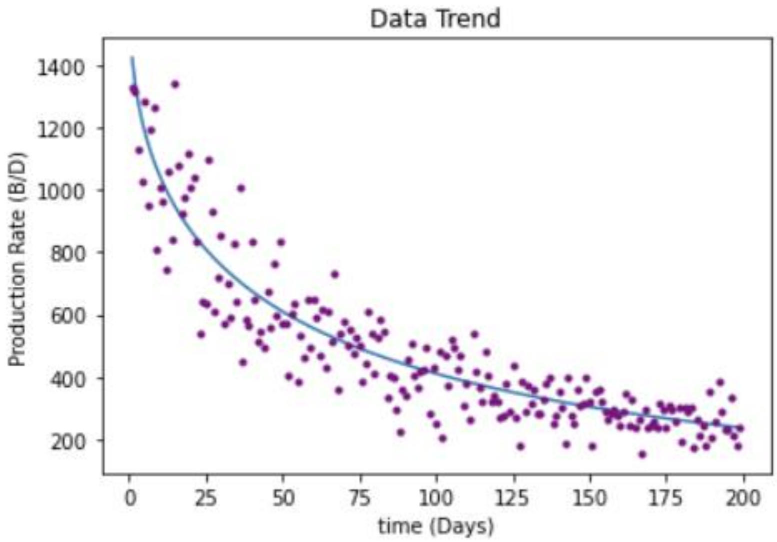


Figure 4-2 Trend of synthetic data averaged

Once the data has been generated, we implement each of the models individually. So, at first, we started with the SEM model. We defined the function and the input parameters of days, The initial flow rate, and n is the number of days and the constant τ. After this, we define our model. Using some initial conditions, some boundaries of both the lower and the upper and this generates. The best possible values. For our given parameters.

Once the model has run via our set parameters, which we then incorporate into the SEM model to see how consistent it is with our initial data. The generated data is shown in the figure below against the generated data that we have.

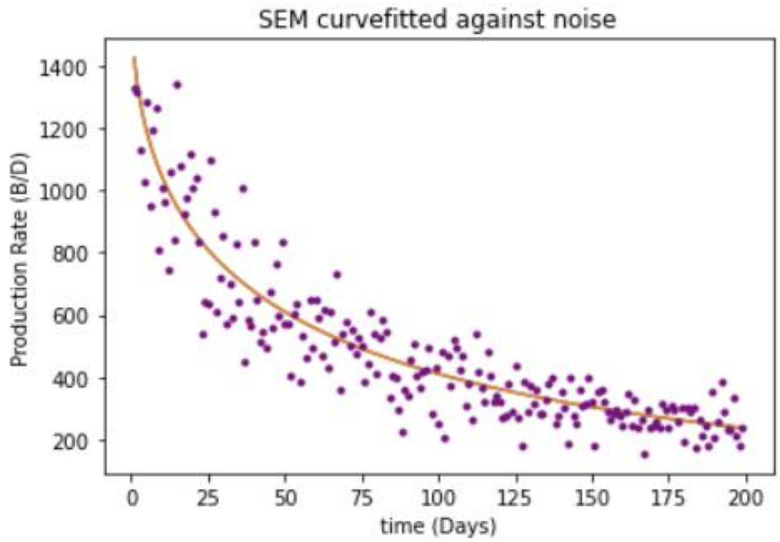


Figure 4-3 SEM against synthetic data

The next step is to set up the model for Arpes. This follows a similar sequence where the function is defined with the relevant input parameters. In this case we use the number of days as time, the initial flow rate for Arpes as q_0, the constant b which is taken to be the decline exponent and D_i being the initial decline exponent.

This data is run against a curve fit fiction which again determines the best parameters and returns them respectively which we use as the input parameters to our model. The resultant data is collected and plotted against the original data as shown in figure 4.4 below:

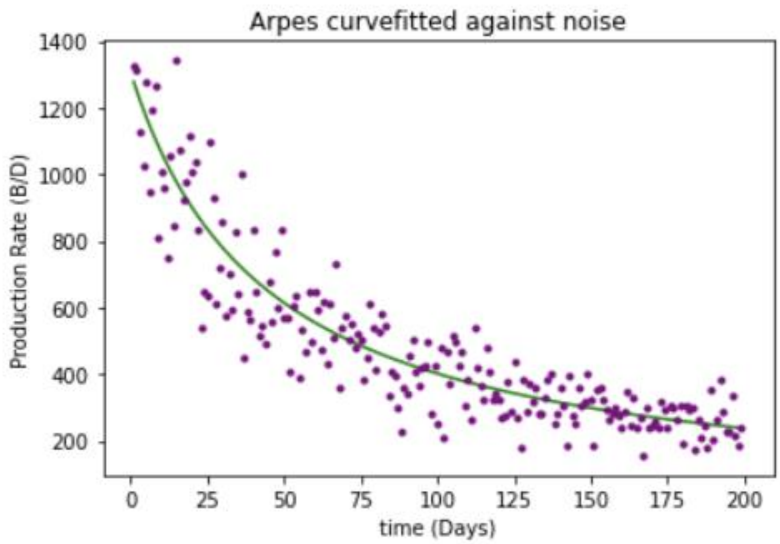


Figure 4-4 Arpes against synthetic data

The next model is the LGM model. Similar to as before, the function is defined with the input parameters that for this model, are the constant a, K being the carrying capacity and n being the hyperbolic exponent.

When running them through the curve fit function, the optimal values are returned and then are used as part of the LGM function that eventually gives the result below in figure 4.5

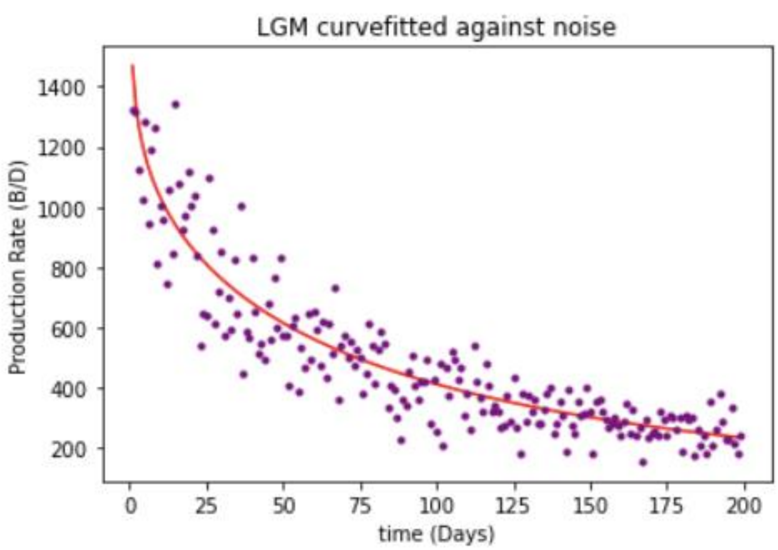


Figure 4-5 LGM against synthetic data

For Pan CRM, we repeat our process except this time instead of three parameters we now use four for the function input. These parameters are J as the productivity, Beta as the linear transient flow parameter, J_∞ as the constant productivity index and c_pV_t as the compressibility and drainage pour volume.

Obtaining the parameters and running the model function results in the graph in the figure below:

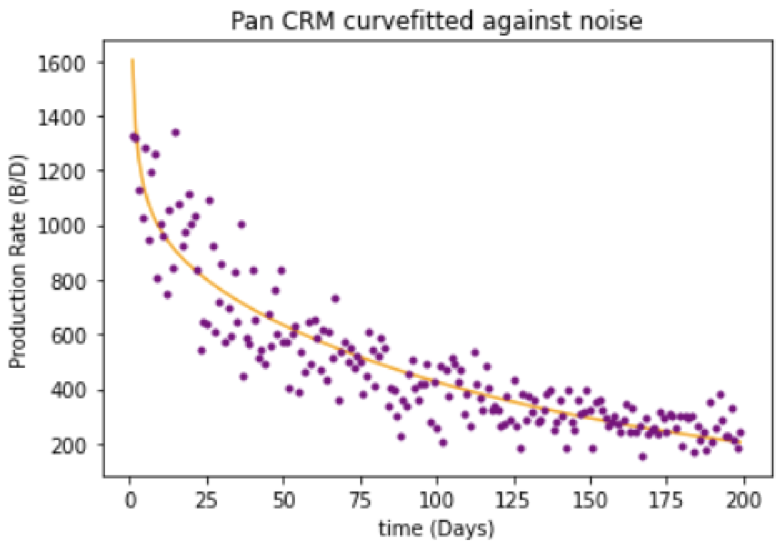


Figure 4-6 Pan CRM against synthetic data

Plotting all the models side by side gives us an overall idea of how the trend is supposed to look like and we can see the consistency between the four models and the data.

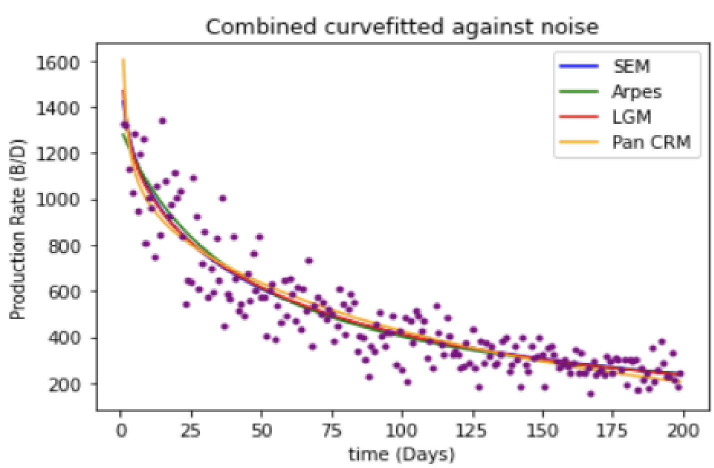


Figure 4-7 All models against synthetic data

Our next step is to implement the moving window algorithm. After collecting the data from running the model functions, all four trends are recorded and transformed into the form of a data frame. This gives us a more streamlined representation of our progress so far.

	Time	Data	SEM	Arpes	LGM	Pan CRM
0	1	1325.756795	1422.019697	1277.806904	1468.089913	1603.567849
1	2	1316.030911	1338.714486	1250.217466	1344.393256	1358.452113
2	3	1125.887910	1278.748490	1223.794228	1270.830626	1245.434949
3	4	1027.436222	1230.611765	1198.464779	1217.245444	1175.100383
4	5	1280.593329	1189.910052	1174.162579	1174.501664	1124.893840
...	...	...	...	...	...	...
194	195	229.365849	242.487973	241.956824	238.733914	210.817059
195	196	332.696088	241.345084	240.949992	237.582187	209.293021
196	197	215.593019	240.210587	239.951505	236.439400	207.780386
197	198	182.759147	239.084394	238.961259	235.305453	206.279061
198	199	240.979393	237.966413	237.979153	234.180250	204.788955

Figure 4-8 Model data collected into a single data frame

Upon representing our data, we add newer columns which come from the result of implementing the moving window function and obtaining the rolling average and the rolling standard deviation. For the moving average implementation our window size is set to 5 and similarly we use the same dimension for the standard deviation. This at the end, will give us the respective column value in the data frame. However, the number of values will be reduced by 4 as the moving window can only be implemented on values greater than or equal to its size.

	Time	Data	SEM	Arpes	LGM	Pan CRM	MovingAverage	Moving SD
0	1	1325.756795	1422.019697	1277.806904	1468.089913	1603.567849	NaN	NaN
1	2	1316.030911	1338.714486	1250.217466	1344.393256	1358.452113	NaN	NaN
2	3	1125.887910	1278.748490	1223.794228	1270.830626	1245.434949	NaN	NaN
3	4	1027.436222	1230.611765	1198.464779	1217.245444	1175.100383	NaN	NaN
4	5	1280.593329	1189.910052	1174.162579	1174.501664	1124.893840	1215.141034	132.190992
...	...	...	...	...	...	...	...	...
194	195	229.365849	242.487973	241.956824	238.733914	210.817059	278.178070	63.636606
195	196	332.696088	241.345084	240.949992	237.582187	209.293021	292.595663	66.750743
196	197	215.593019	240.210587	239.951505	236.439400	207.780386	259.052305	49.742371
197	198	182.759147	239.084394	238.961259	235.305453	206.279061	238.006205	56.271214
198	199	240.979393	237.966413	237.979153	234.180250	204.788955	240.278699	56.076815

Figure 4-9 Obtained moving window average and standard deviation

On obtaining these two values, we graph the results with the original values, and we can see that the average and standard deviation are both consistent with our data.

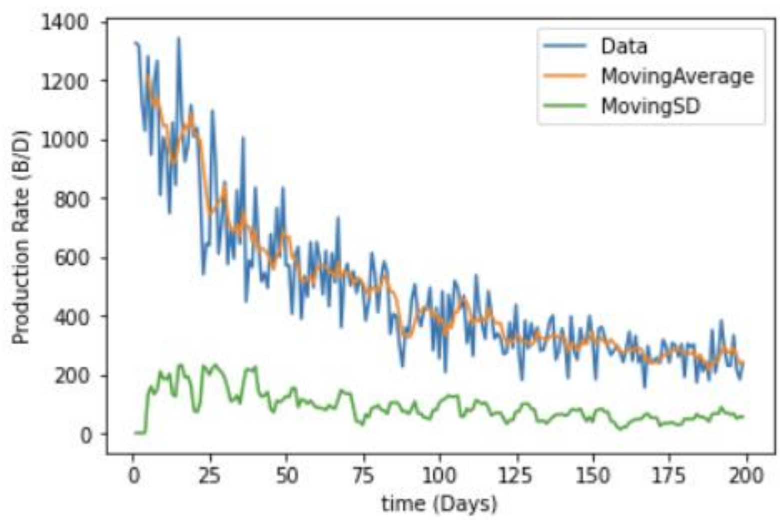


Figure 4-10 Plotting moving window values against our data

Our next step is to use the standard deviation and average to randomly generate a set number of samples. In our case, we have used the sample size of 10 for testing the synthetic data.
On generating these curves, our next step is to rank the highest values of all the samples as one curve. We repeat the same step with all the second highest values of each time step and same for the third and so on until all the generated samples are ranked according to their respective order

of maximum value. Our resulting data frame will give us all columns arranged in a descending order.

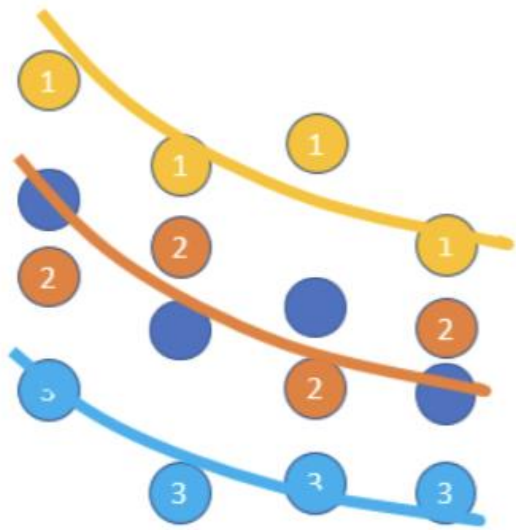


Figure 4-11 Ranking samples from highest to lowest

The ranked distribution of the decline curves can be shown in figure 4.12 below.

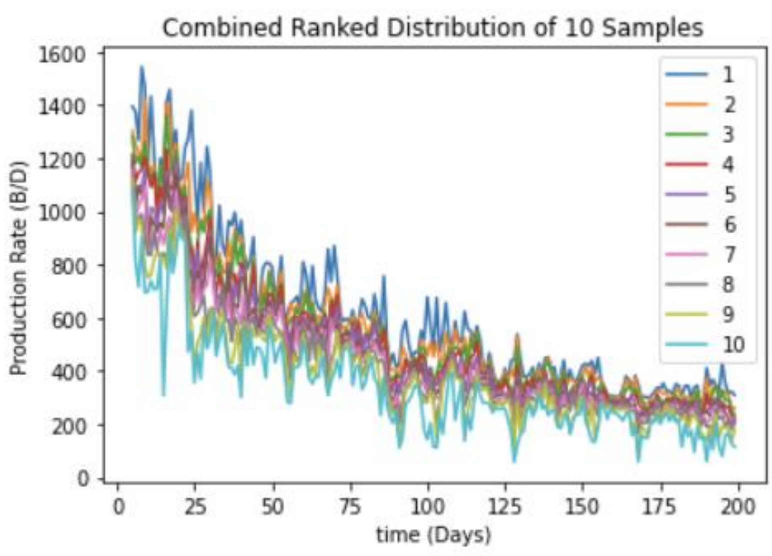


Figure 4-12 Plotting moving window values against our data

All these samples are then made into a new data frame along with their timestep and moving standard deviation.

	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	t
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	1
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	2
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	4
4	1394.403746	1302.563270	1280.342731	1211.958270	1192.913167	1188.353205	1154.155774	1132.258456	1123.682142	1086.868555	5
...	...	...	...	...	...	...	...	...	...	...	...
194	433.982047	327.675824	325.660492	281.599848	276.027865	275.348003	246.264830	228.685884	189.723792	79.284072	195
195	352.421951	319.649051	289.511673	267.885570	243.510129	234.967127	229.308401	188.963597	187.354137	158.264946	196
196	316.196638	310.739573	294.165164	263.033526	225.054306	220.575451	217.740162	214.170522	181.843216	166.135945	197
197	322.417280	258.843489	253.305831	232.668058	206.824870	196.088471	189.790228	179.157583	133.388357	126.814010	198
198	309.304494	261.307862	235.858113	220.176791	218.491063	209.576969	206.121446	204.133960	180.462368	114.653097	199

199 rows × 11 columns

Figure 4-13 Samples converted to a data frame

From the above figure, we can see that the first four values appear as NaN. This is due to the unavailability of their respective standard deviations hence these values need to be removed. After removing them our data points are now down to 196 instead of the original 200.

	S0	S1	S2	S3	S4	S5	S6	S7	S8	S9	t
4	1394.403746	1302.563270	1280.342731	1211.958270	1192.913167	1188.353205	1154.155774	1132.258456	1123.682142	1086.868555	5
5	1375.059054	1253.963645	1181.568417	1113.383947	1053.128788	1015.483555	975.218011	930.577930	921.997880	814.208409	6
6	1267.524844	1223.131744	1206.777008	1146.642422	1098.877678	1090.406297	1019.704001	976.616418	970.479318	717.061439	7
7	1543.570947	1169.507655	1166.159287	1157.776978	1085.598581	1062.491185	1048.839520	990.206343	948.588084	903.408025	8
8	1465.432441	1423.105977	1261.954757	1203.136812	1166.975553	1098.623933	1096.064733	903.744541	793.362058	695.189657	9
...	...	...	...	...	...	...	...	...	...	...	...
194	433.982047	327.675824	325.660492	281.599848	276.027865	275.348003	246.264830	228.685884	189.723792	79.284072	195
195	352.421951	319.649051	289.511673	267.885570	243.510129	234.967127	229.308401	188.963597	187.354137	158.264946	196
196	316.196638	310.739573	294.165164	263.033526	225.054306	220.575451	217.740162	214.170522	181.843216	166.135945	197
197	322.417280	258.843489	253.305831	232.668058	206.824870	196.088471	189.790228	179.157583	133.388357	126.814010	198
198	309.304494	261.307862	235.858113	220.176791	218.491063	209.576969	206.121446	204.133960	180.462368	114.653097	199

195 rows × 11 columns

Figure 4-14 Removed NaN values

With the data cleaned and ready to be run through our models, we implement each of the models on the given data to obtain all the possible decline curves.

Each ranked curved is passed through the curve fit function. This time we use and additional parameter of sigma in the curve fit which is equal to the moving standard deviation. This allows us to keep it as a weighted average in our probability calculations and then the resultant parameters are passed through the model function and are graphed respectively.

We can see the result of passing the data through each model in the figures below.

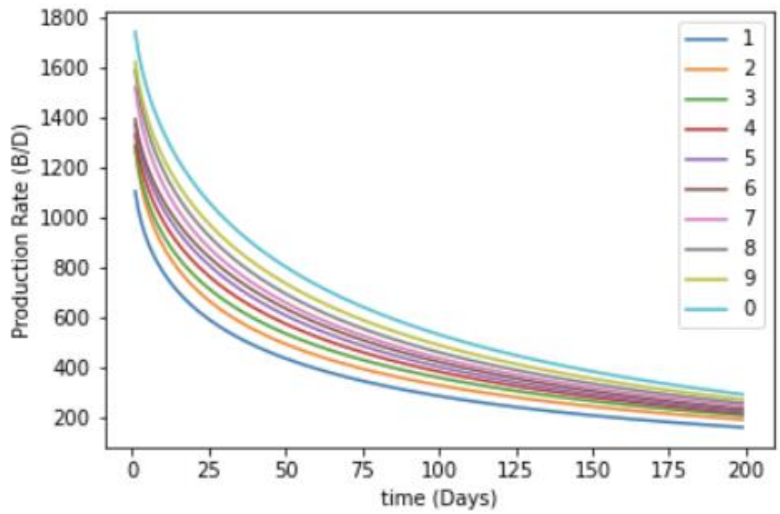


Figure 4-15 Combined ranked SEM Models

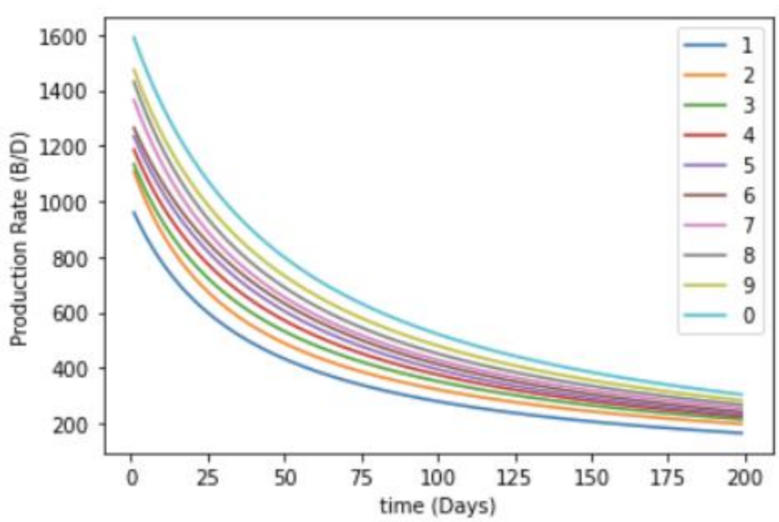


Figure 4-16 Combined ranked Arpes Models

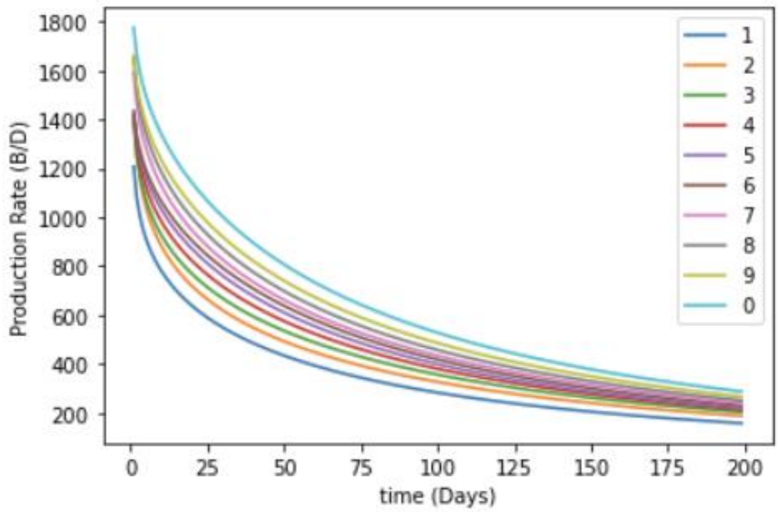


Figure 4-17 Combined ranked LGM Models

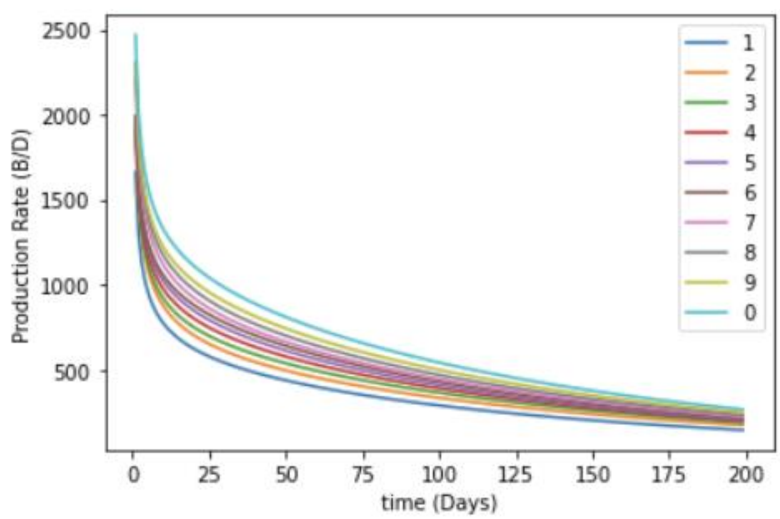


Figure 4-18 Combined ranked Pan CRM Models

After running the simulation for all the ranked decline curves. Our next step is to set up the maximum likelihood function.

The maximum likelihood function takes the squared difference between the original data and the simulated decline curve divided by the squared standard deviation at the specific time step. All the datapoints from each timestep are then added together to attain the maximum likelihood.

$$L_{MLE} = \sum_{K=1}^{T} \frac{[q_k(x) - \hat{q}_k]^2}{{\sigma_k}^2}$$

(10)

This is done for the four models and we at the end are with arrays for all the models.

For the probability calculation, each index of the data frame is compared and computed to get the probability for each model.

The probabilities are then ranked with the higher the probability the more likely the best model.

Model	Probability
SEM	0.014859
Arpes	0.115539
LGM	0.001562
Pan CRM	0.868040

Table 4.1 Final probabilities

Using AIC and Corrected AIC

Our next approach utilizes AIC. For the first step, we calculate the sum of squares and relate it to the parameters of AIC. These parameters are N which is the number of data points used in the model, ss which is the squared sum that we already calculated, and K is the number of parameters used in the model plus one.

Using the results from AIC, we use the value to calculate the Corrected AIC Or AIC_c which was already stated to be more accurate than the regular AIC calculation.

According to the definition of AIC, the lower the value, the better the fit.

On collecting and rearranging our data as a separate data frame to be used, the squared sum for each model was calculated followed by the AIC calculation until we get the resulting data frame column for AIC.

On calculating the squared sum and implementing the AIC equation, we obtain the result below:

	Model	Raw_Noise	Data	Squared Sum	AIC
0	sem	[1501.441441869447, 1475.8167680307627, 1258.3...	[465.50695511309925, 461.54767961919094, 457.7...	1.919571e+07	2293.901637
1	arpes	[1501.441441869447, 1475.8167680307627, 1258.3...	[456.33647438944803, 456.33647438944803, 456.3...	1.928353e+07	2294.809993
2	lgm	[1501.441441869447, 1475.8167680307627, 1258.3...	[502.9875331149792, 497.81586968664584, 492.72...	1.849162e+07	2286.465143
3	panCRM	[1501.441441869447, 1475.8167680307627, 1258.3...	[1603.5678494692888, 1358.45211250099, 1245.43...	2.885319e+06	1918.786370

Figure 4-19 Squared sum and AIC results

To attain more accuracy, we now switch over to AIC_c which is done so using the calculated AIC values. On doing so we now obtain a new column for the Corrected AIC values.

	Model	Raw_Noise	Data	Squared Sum	AIC	CAIC
0	sem	[1501.441441869447, 1475.8167680307627, 1258.3...	[465.50695511309925, 461.54767961919094, 457.7...	1.919571e+07	2293.901637	2294.212517
1	arpes	[1501.441441869447, 1475.8167680307627, 1258.3...	[456.33647438944803, 456.33647438944803, 456.3...	1.928353e+07	2294.809993	2295.120873
2	lgm	[1501.441441869447, 1475.8167680307627, 1258.3...	[502.9875331149792, 497.81586968664584, 492.72...	1.849162e+07	2286.465143	2286.776024
3	panCRM	[1501.441441869447, 1475.8167680307627, 1258.3...	[1603.5678494692888, 1358.45211250099, 1245.43...	2.885319e+06	1918.786370	1919.223870

Figure 4-20 AIC and corrected AIC results

As mentioned, this table now as to be sorted in an ascending order for both AIC and AIC_c

	Model	Raw_Noise	Data	Squared Sum	AIC	CAIC
3	panCRM	[1501.441441869447, 1475.8167680307627, 1258.3...	[1603.5678494692888, 1358.45211250099, 1245.43...	2.885319e+06	1918.786370	1919.223870
2	lgm	[1501.441441869447, 1475.8167680307627, 1258.3...	[502.9875331149792, 497.81586968664584, 492.72...	1.849162e+07	2286.465143	2286.776024
0	sem	[1501.441441869447, 1475.8167680307627, 1258.3...	[465.50695511309925, 461.54767961919094, 457.7...	1.919571e+07	2293.901637	2294.212517
1	arpes	[1501.441441869447, 1475.8167680307627, 1258.3...	[456.33647438944803, 456.33647438944803, 456.3...	1.928353e+07	2294.809993	2295.120873

Figure 4-21 AIC and corrected AIC ranked

From the results above we can see that there is no difference in the rankings when using AIC or the corrected AIC.

In comparison to the probabilistic approach, the results are also found to be consistent for Pan CRM and SEM but not for Arpes and LGM.

5. Study with real data

In this chapter, for the purposes of illustration, data from well 41 is demonstrated in the figures and tables.

Data requirement

Similar to the previous section, values for the parameters for the models are required, however, the curve fit function with a set of initial parameters and a starting point can help us determine the appropriate parameters.

Well data from 26 different wells that were recorded over time were used as the experimental data. This data was presented in the form of a Microsoft Excel sheet. This data was imported on a well-by-well basis and was run through the program that was constructed for the synthetic data.

Our code had to be modified to accommodate the different lengths of the input data.

The first process was to import the excel file and for that purpose, we used the pd.io.excel.read_excel() function. This allows us to access the file so that we can read the well data.

Sheet 1 on the excel file gives us a detailed explanation of column values associated with the well. It is therefore important we single out the information we need.

The relevant columns of time in days and the oil rate in barrels per day were extracted while all other information was discarded.

The next step involved reading the data. In order to understand and get a better visual representation of our excel sheet, the data is imported as a data frame which indicates the number of rows and columns

For our calculations, we will only consider the time (t_mid.1) and the extraction rate (Oil Rate [bbl/day].1) as our parameters. On obtaining this information, our data frame will look as shown below.

```
0       0.000000
1       0.000000
2       0.000000
3       0.000000
4       0.000000
          ...
622    19.700000
623    16.387097
624    19.774194
625    15.535714
626    17.032258
Name: Oil Rate [bbl/day].1, Length: 627, dtype: float64
```

Figure 5-1 Imported well data

On inspecting this table, we notice one aspect in the data that the extraction values for a considerable part of our data set is zero which implies extraction has not started yet. It is therefore important to have our decline curve as smooth as possible, so we drop the initial zero values and relabel the index.

The final step in data pre-processing involves resetting the time values in accordance to starting at days the extraction process has begun. For this we subtract the first-time index from the rest of the array and we get our final form of our real word data set to be processed.

	t_mid.1	Oil Rate [bbl/day].1
0	0.0	49.548387
1	30.5	313.100000
2	61.0	324.096774
3	92.0	291.967742
4	122.5	253.866667
...	...	...
90	2740.5	19.700000
91	2771.0	16.387097
92	2802.0	19.774194
93	2831.5	15.535714
94	2861.0	17.032258

95 rows × 2 columns

Figure 5-2 Cleaned data

Graphing the data, we can see it looks similar to our synthetic data that we generated previously.

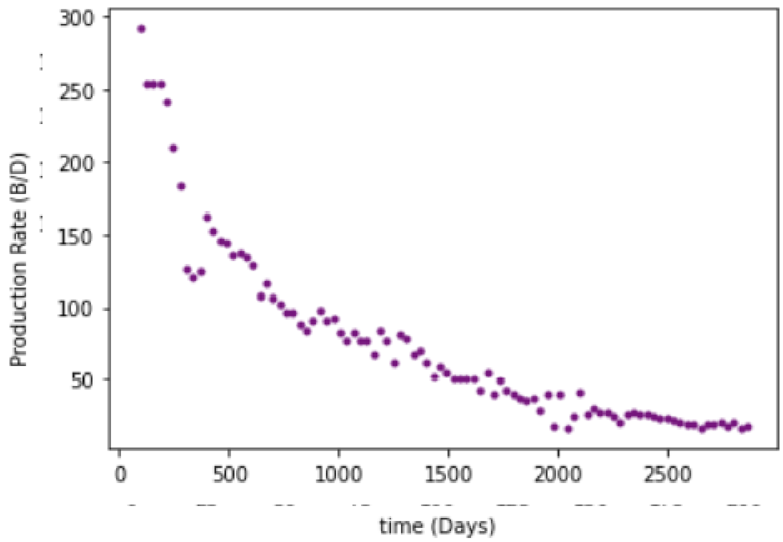


Figure 5-3 plotted well data

This data is now ready to be run through our four models and doing so would result in the following plots for each model:

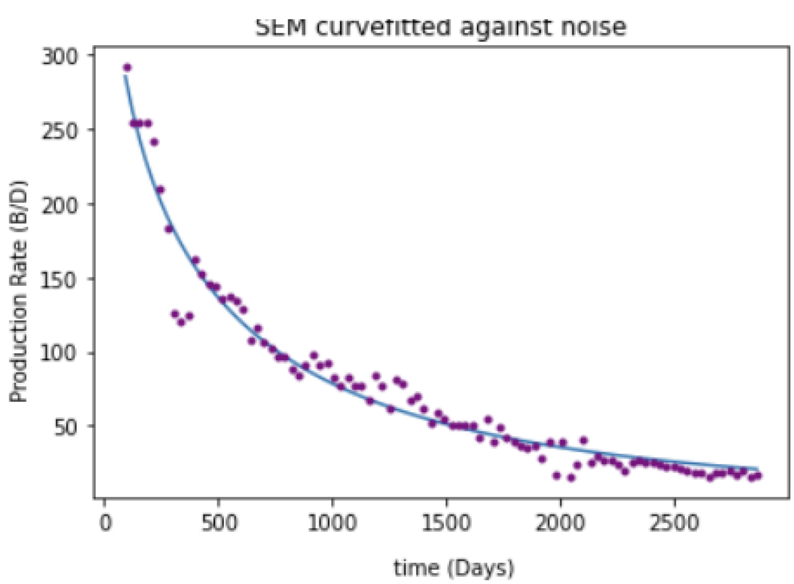


Figure 5-4 plotted SEM model against well data

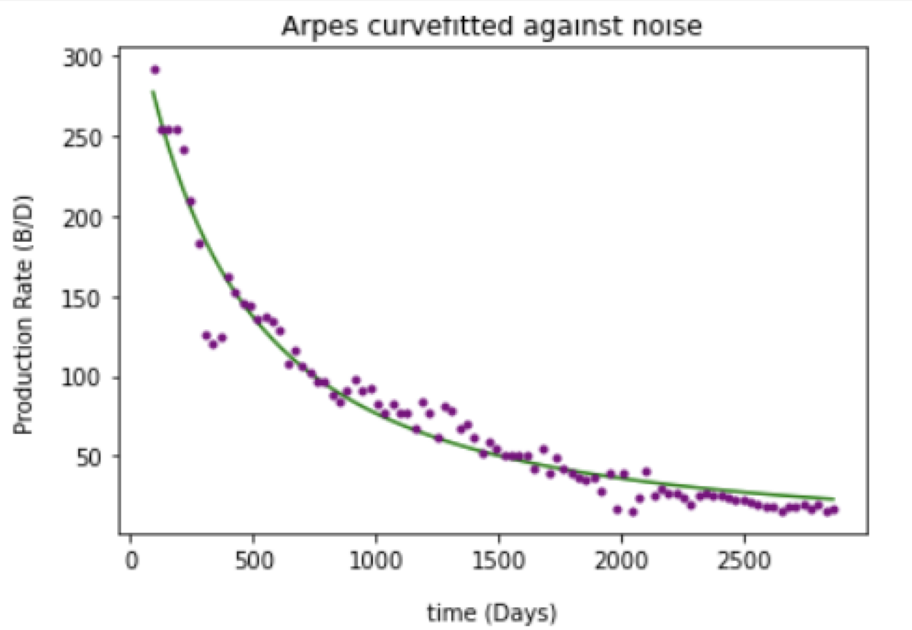


Figure 5-5 plotted Arpes model against well data

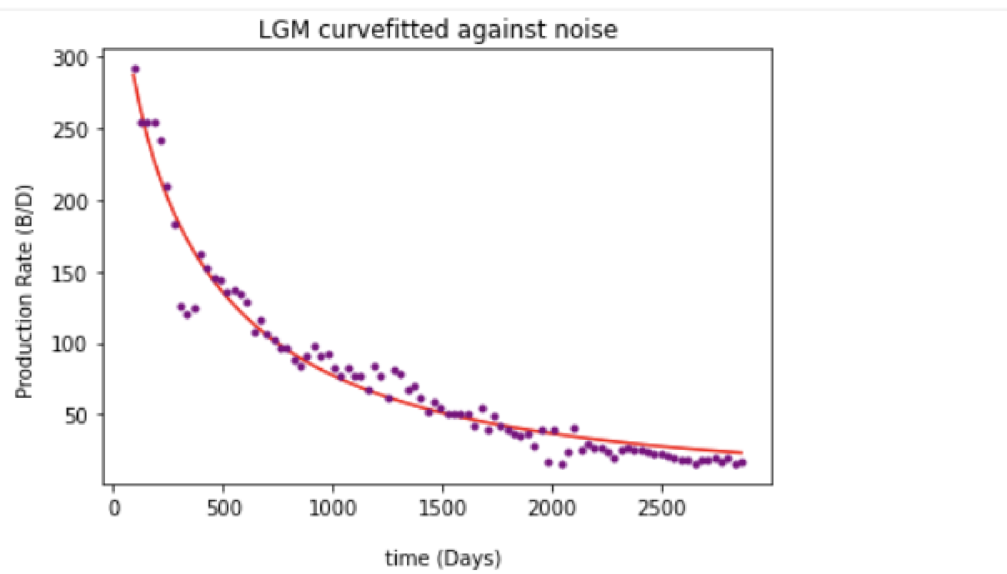


Figure 5-6 plotted LGM model against well data

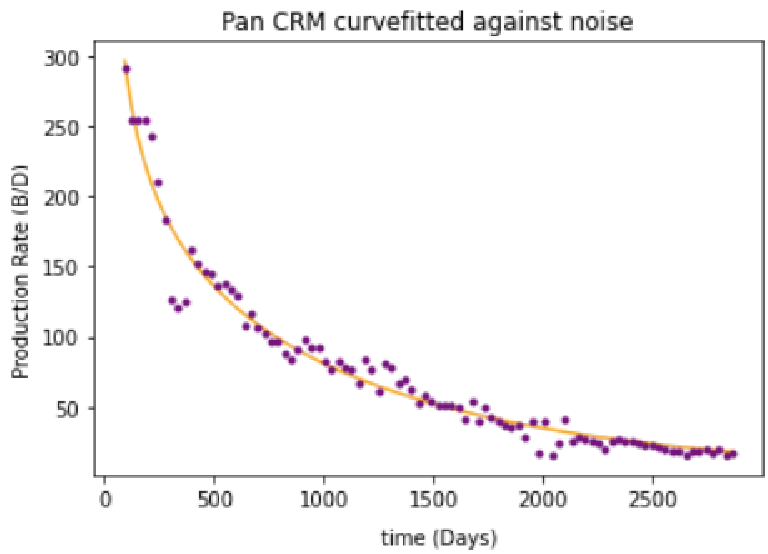


Figure 5-7 plotted Pan CRM model against well data

The next step is to implement our moving window function to obtain an average and a standard deviation. Just like in our synthetic data testing, we keep the moving window size to 5. This is processed and results in the graph representation below:

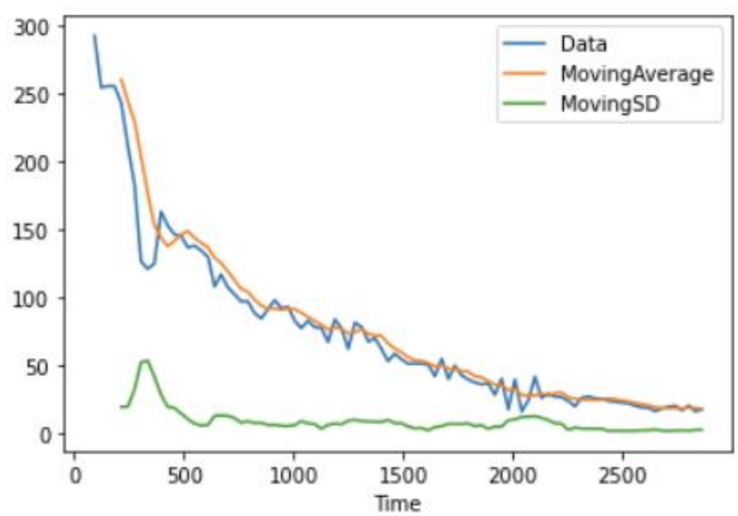


Figure 5-8 Implementation of moving window

On obtaining the average and rolling standard deviation, one thing to not is the existence of NaN values. NaN referrers to "Not a Number" which is to be expected as the moving window only gives values of indexes greater than the window size and onwards.

On removing the first four NaN rows we notice that there are still some indexes where the standards deviation is NaN. This may possibly be a result of the moving window function limitation. To keep our data consistent, these rows are removed as well.

	Time	Data	SEM	Arpes	LGM	Pan CRM	MovingAverage	MovingSD
3	92.0	291.967742	285.702053	277.650886	287.374863	296.666883	NaN	NaN
4	122.5	253.866667	262.733371	260.101222	264.062645	264.684632	NaN	NaN
5	153.0	255.000000	243.902723	244.287684	244.981736	241.710738	NaN	NaN
6	183.5	255.000000	227.956298	229.981072	228.793010	223.911681	NaN	NaN
7	214.0	242.548387	214.152049	216.989494	214.741063	209.431827	259.676559	18.798870
...	...	...	...	...	...	...	...	...
90	2740.5	19.700000	22.090287	24.119646	24.456039	19.537717	18.028602	1.481212
91	2771.0	16.387097	21.696977	23.776492	24.098646	19.090249	17.680215	1.647236
92	2802.0	19.774194	21.306435	23.435780	23.743656	18.646547	18.486667	1.434359
93	2831.5	15.535714	20.943167	23.118869	23.413343	18.234431	18.047143	1.962412
94	2861.0	17.032258	20.587835	22.808874	23.090125	17.831926	17.685853	1.946471

Figure 5-9 Existence of more NaN values which are also removed

We are now ready to generate samples of our data. This is done so by using the by using the obtained standard deviation.

This code was tested using samples of various sizes with the maximum being 100,000. But after comparisons the difference in our final probability results between sample sizes of 100,000 and 10,000 were found to be minimal. So in this thesis, and for all our experimental wells, the sample size we will continue to use sizes of 10,000.

Once our samples are generated, we use a sorting function to sort the maximum values of each time step as one curve, the second maximum values as another and so on until all the curves are sorted and ranked in order of their highest values.

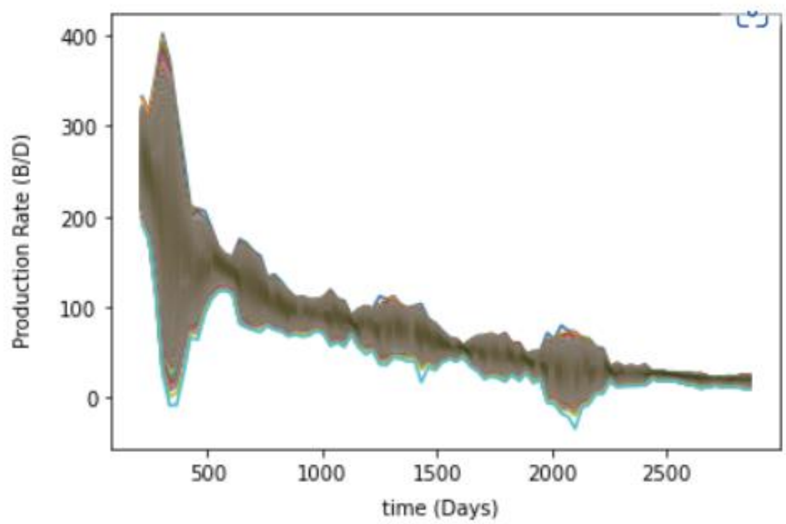


Figure 5-10 Visual ranking of 10,000 samples

All these samples are brought into a data frame and the data frame is appended with the respective time and standard deviation values.

We remove any final NaN values and after doing so, our data is ready for processing.

6	7	8	9	...	9992	9993	9994	9995	9996	9997	9998	9999	t	MovingSD
369.528463	368.258964	366.526729	364.472995	...	41.988108	39.541436	37.434743	33.936957	32.553713	28.984449	26.743249	26.552042	214.0	18.798870
355.616623	350.294883	345.785681	344.958396	...	19.274093	17.512056	14.646875	13.678205	11.211448	7.557792	0.883697	-8.384098	245.0	19.262478
287.196910	287.179428	286.820662	284.148686	...	26.894037	20.533529	17.754638	17.556619	16.178524	13.534856	5.823673	-7.894625	274.5	31.726042
231.355198	229.022037	228.724486	228.643647	...	57.446871	57.414388	57.212031	51.993972	50.731718	50.053515	31.385381	26.431244	304.0	51.651471
195.265564	194.809351	193.675343	193.021046	...	79.173110	78.794464	77.450111	75.927274	75.361871	74.769184	71.669345	66.737959	334.5	52.996862
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
22.780610	22.664896	22.661585	22.618164	...	13.406445	13.240243	13.131391	13.131234	13.080413	13.042265	12.859583	12.707942	2649.0	2.117634
22.966927	22.949025	22.876882	22.842033	...	12.365829	12.166569	11.937670	11.870202	11.808909	11.599524	11.593480	11.440540	2679.5	1.497629
23.163170	23.136922	23.062774	23.042019	...	13.983341	13.934053	13.852555	13.828190	13.456458	13.239968	13.214492	12.993750	2710.0	1.221159
24.245795	24.196162	24.182817	24.147271	...	11.952840	11.947148	11.678285	11.640802	11.512997	11.471318	11.323607	9.732010	2740.5	1.481212
23.874951	23.857205	23.837951	23.672372	...	11.208437	11.205666	10.891528	10.795580	10.612729	9.995783	9.709718	9.513053	2771.0	1.647236

Figure 5-11 Removal of all NaN values

Each model is subsequently run on all the sampled decline curves and our resulting plots are generated. The below figures visualize the obtained decline curves from running on the data from well 41.

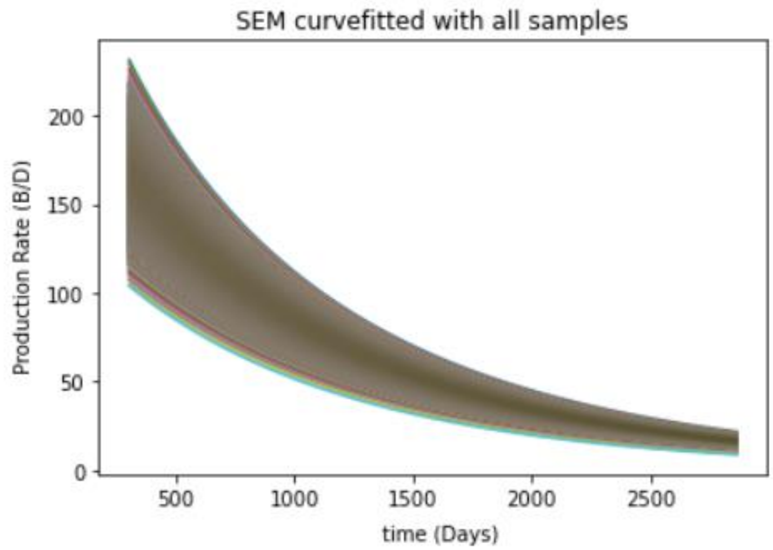


Figure 5-12 Curve fit SEM 10,000 samples

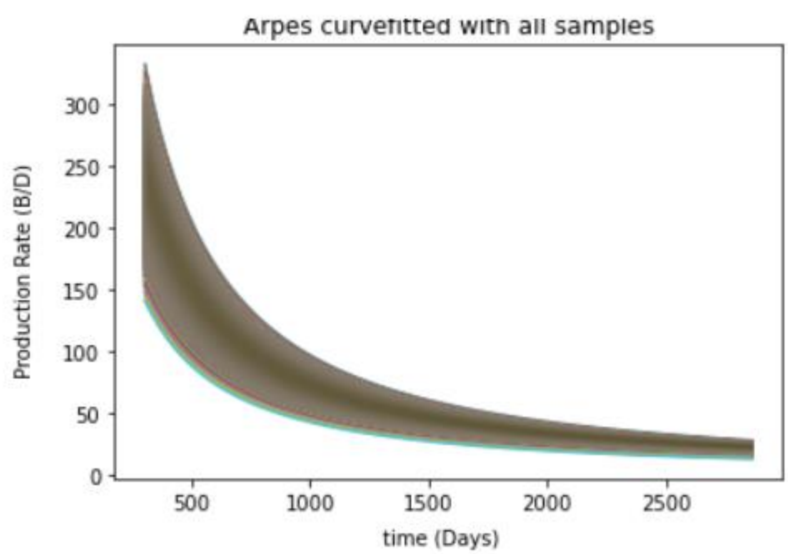


Figure 5-13 Curve fit Arpes 10,000 samples

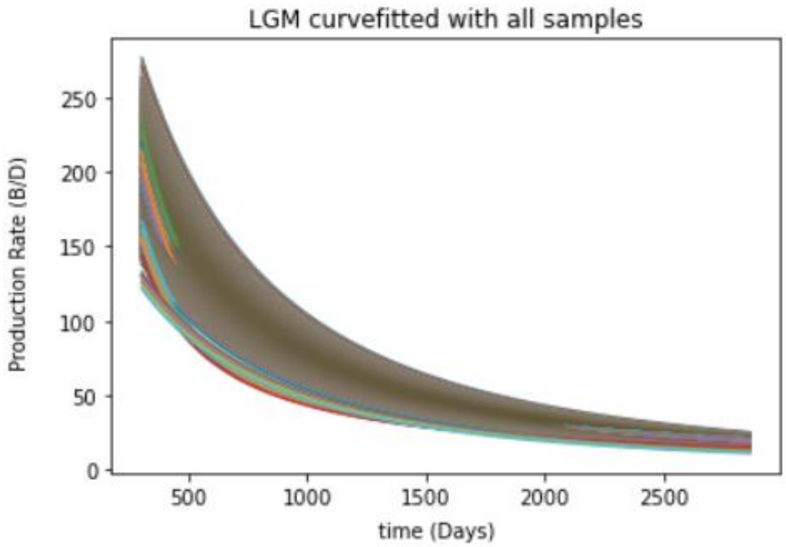


Figure 5-14 Curve fit LGM 10,000 samples

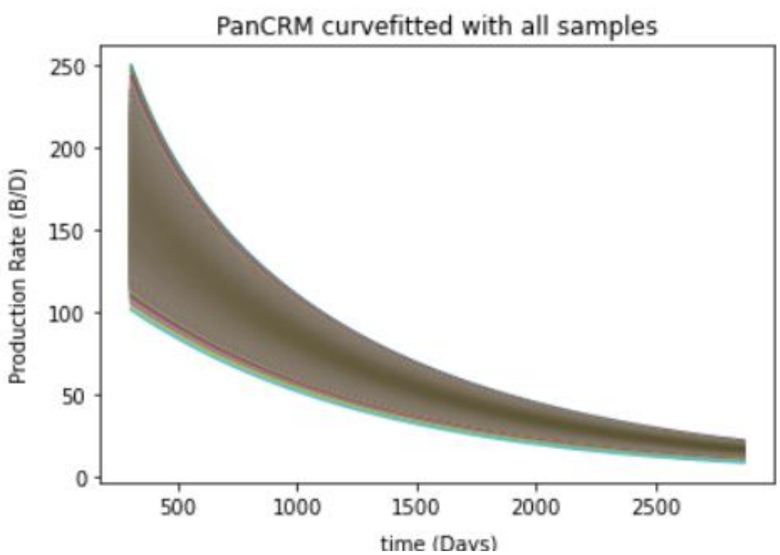


Figure 5-15 Curve fit Pan CRM 10,000 samples

Our next step involves the maximum likelihood calculation. This is calculated by having our original data subtracted by our generated data and the result being squared and divided by the standard deviation squared as given in the equation 10

On doing so for each model and its data, our result is saved into another data frame.

Once we have our data frame for maximum likelihoods, our next step involves calculating the final probabilities. This is done by the equation 11 where all the likelihoods are added up

On calculating the individual probabilities, we compile our results in the form of a data frame which shows the individual probabilities of each model against each other for each sample.

	SEM	Arpes	LGM	Pan CRM
0	0.000059	1.013183e-46	1.244911e-22	0.000041
1	0.000066	2.181235e-47	9.199780e-23	0.000034
2	0.000070	4.587958e-48	5.616585e-23	0.000030
3	0.000069	1.551681e-48	3.307390e-23	0.000031
4	0.000067	5.428022e-49	1.989203e-23	0.000033
...	...	...	...	...
9995	0.000026	1.123921e-71	4.024498e-26	0.000074
9996	0.000024	1.155734e-71	4.831206e-26	0.000076
9997	0.000022	1.271815e-71	6.575176e-26	0.000078
9998	0.000028	6.908083e-71	2.464308e-25	0.000072
9999	0.000026	2.857933e-68	1.563089e-23	0.000074

10000 rows × 4 columns

Figure 5-16 **Maximum likelihood calculations**

To calculate the final probability, the columns are added, and the obtained value signifies the final probability of that model being applicable on the tested data set.

SEM	6.082972e-01
Arpes	1.304238e-46
LGM	4.825889e-22
Pan CRM	3.917028e-01

Table 5.1 Final probabilities

On moving to the AIC calculation, there are not as many modifications to be made to accommodate the well data set. The only calculation needed to be made is to obtain the length of the dataset i.e. the time variable. This is done simply by using the len() function on the oil rate data that was cleaned to start from the time of production.

Once this is obtained, the relevant variables are put into the AIC equation (Equation 8) and for each model the respective AIC is obtained.

Since (Shabib-Asl & Plaksina, 2019) said the Corrected AIC yielded better results, this thesis looked into implementing that as well.

The calculation for the Corrected AIC is also relatively straightforward. It makes use of our previous AIC calculation for a newer calculation.

This yields another set of results, that are different in values but similar to the AIC calculations.

On comparing both models over various wells, the following results were obtained.

	Model	Raw_Noise	Data	Squared Sum	AIC	CAIC
3	panCRM	10 125.903226 11 120.466667 12 124.41...	3 296.666883 4 264.684632 5 241.71...	10117.554275	418.246959	419.323882
0	sem	10 125.903226 11 120.466667 12 124.41...	10 104.052342 11 100.777758 12 97.62...	67073.308187	577.025667	577.785160
2	lgm	10 125.903226 11 120.466667 12 124.41...	10 121.517931 11 115.235070 12 109.42...	70680.336969	581.478072	582.237566
1	arpes	10 125.903226 11 120.466667 12 124.41...	10 140.723032 11 129.172995 12 119.22...	79378.642364	591.343336	592.102830

Figure 5-17 **Ranked AIC and Corrected AIC values**

6. Results

Having compared two different models over 13 different wells

The following results were obtained after running the probabilistic model for a sample size of 10,000.

Well ID	Probabilistic		Corrected AIC	
9	LGM	0.87	Pan CRM	157.5
	SEM	0.1	Arpes	313
	Pan CRM	0.01	LGM	333.8
	Arpes	0	SEM	334.2
11	Arpes	0.27	Pan CRM	322.8
	Pan CRM	0.25	SEM	458.5
	LGM	0.24	Arpes	458.58
	SEM	0.21	LGM	464.72
13	Pan CRM	4.91E-01	Pan CRM	434.1
	SEM	4.91E-01	SEM	585.4
	Arpes	1.77E-02	Arpes	597.9
	LGM	3.88E-07	LGM	621.1
17	Arpes	0.75	Pan CRM	423.5
	SEM	0.18	Arpes	570.9
	LGM	0.05	SEM	572.1
	Pan CRM	0.005	LGM	576.4
18	LGM	0.58	Pan CRM	117.3
	SEM	0.41	LGM	219.8
	Arpes	0.00287	SEM	223
	Pan CRM	0.00281	Arpes	223.9
19	Pan CRM	7.76E-01	Pan CRM	289.2
	SEM	2.23E-01	Arpes	420

	LGM	3.47E-05	SEM	445.5
	Arpes	8.05E-37	LGM	448.5
	pan CRM	6.64E-01	Pan CRM	423.2
	Arpes	3.52E-01	Arpes	482.3
	SEM	7.53E-08	LGM	532.8
22	LGM	5.96E-29	SEM	537.1
	Pan CRM	0.42	Pan CRM	186.2
	SEM	0.31	Arpes	292.3
	LGM	0.25	LGM	300.2
24	Arpes	0	SEM	300.3
	Pan CRM	0.84	Pan CRM	351.2
	SEM	0.15	Arpes	447.8
	Arpes	0.002	SEM	450.3
25	LGM	0	LGM	451.5
	LGM	0.72	Pan CRM	303.8
	SEM	0.16	SEM	436
	Pan CRM	0.03	LGM	439
32	Arpes	0.07	Arpes	441.6
	Pan CRM	0.936774	Pan CRM	381.395564
	Arpes	0.03238	SEM	549.710574
	SEM	0.030463	Arpes	550.30196
35	LGM	0.000383	LGM	551.232969
	Pan CRM	8.17E-01	Pan CRM	300.264857
	SEM	1.83E-01	SEM	479.237176
	LGM	1.51E-14	Arpes	479.691258
40	Arpes	1.19E-45	LGM	488.236681

	SEM	6.08E-01	Pan CRM	419.323882
	Pan CRM	3.92E-01	SEM	577.78516
41	LGM	4.83E-22	LGM	582.237566

	Arpes	1.30E-46	Arpes	592.10283

Table 6.1 Table of results and ranking

7. Discussion, conclusion, and recommendation for future work

Discussion

The purpose of this thesis was to compare on the two different methodologies and to see the similarities, differences in both, the implementation, the process and at the end, the result.
From our final readings, we can conclude that the two different methodologies result in contrasting results. There may be multiple reasons for this.
From our results, we can see that in the AIC method, Pan CRM has consistently performed as the most preferred model despite it having the most parameters. This is likely due to the freedom a model is likely to exhibit if it has more parameters and hence a better fit may compensate for the increased parameters.
Another observation is of well 13 where the ranking has been consistent across both methods. We can conclude for this well the Pan CRM is the most preferred method.
One advantage of using a probabilistic approach is that it provides weightages which may prove beneficial as it helps determine the model that is more likely to be correct instead of relying on one single model.
One disadvantage of using the probabilistic approach is the number of steps and the complexities involved. Unlike the AIC method where using two equations can yield results without much computation, there are many steps required to conduct the probabilistic approach. This is especially time consuming as the number of samples can increase simulation time. In our case the time taken to simulate the results for 100,000 samples took 12 hours to compute for a single well.

Conclusion

For a conclusion, it can be said that there may be more investigation required to bring tangible conclusions. This experiment has shown that there my be a possibility for over confidence in one particular model through a specific method but another model may deny it completely.

Recommendation for future work

This section is dedicated for future work if anyone decides to dedicate time to this study.

- More well data and testing is required. This study was conducted over 13 wells and a sample size of this size may not prove to be bi enough to yield usable results.
- More data about the type of wells can be useful. The wells used in this study were mostly unconventional wells which explains some behaviors such as in the case of Arpes which tends to overestimate in such wells.
- Compare forecasting with the preferred model from each method.

References

- Hong, A., Bratvold, R. B., Lake, L. W., & Ruiz Maraggi, L. M. (2019). Integrating Model Uncertainty in Probabilistic Decline-Curve Analysis for Unconventional-Oil-Production Forecasting. *SPE Reservoir Evaluation & Engineering*, *22*(03), 861–876. https://doi.org/10.2118/194503-PA
- *Law of population*. (n.d.). 9.
- Luis F., A. H., & Ye, P. (2012). Unified Decline Type-Curve Analysis for Natural Gas Wells in Boundary-Dominated Flow. *SPE Journal*, *18*(01), 97–113. https://doi.org/10.2118/161095-PA
- Poston, S. W., & Poe, B. D. (2007). *Analysis of Production Decline Curves*. Society of Petroleum Engineers. http://ebookcentral.proquest.com/lib/uisbib/detail.action?docID=3404997
- Valkó, P. P. (2009, January 19). *Assigning value to stimulation in the Barnett Shale: A simultaneous analysis of 7000 plus production hystories and well completion records*. SPE Hydraulic Fracturing Technology Conference. https://doi.org/10.2118/119369-MS
- Valkó, Peter P., and W. John Lee. "A Better Way to Forecast Production from Unconventional Gas Wells." Paper presented at the SPE Annual Technical Conference and Exhibition, Florence, Italy, September 2010. doi: https://doi.org/10.2118/134231-MS

    - Tadjer A, Hong A, Bratvold RB. Machine learning based decline curve analysis for short-term oil production forecast. Energy Exploration & Exploitation. 2021;39(5):1747-1769. doi:10.1177/01445987211011784

- Can, B., & Kabir, C. S. (2011, June). Probabilistic performance forecasting for unconventional reservoirs with stretched-exponential model. In *North American Unconventional Gas Conference and Exhibition*. OnePetro.

- Box, G. E. (1979). Robustness in the strategy of scientific model building. In *Robustness in statistics* (pp. 201-236). Academic Press.

    - Pan, Z. (2016). Revised productivity index equation to improve transient history match for the capacitance resistance model (Doctoral dissertation).
- Johnson, R.H., Bollens, A.L., 1927. The loss ratio method of extrapolating oil well decline curves. Trans. AIME 77 (01), 771–778. https://doi.org/10.2118/927771-G.

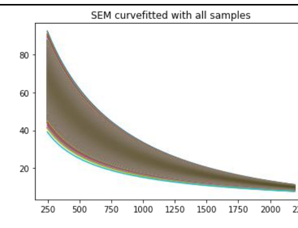
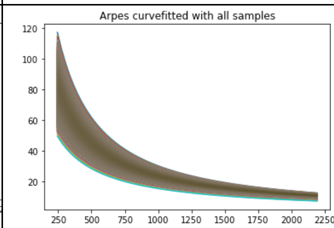
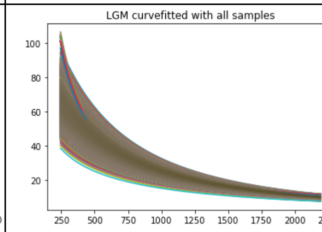
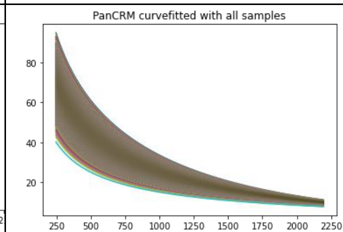
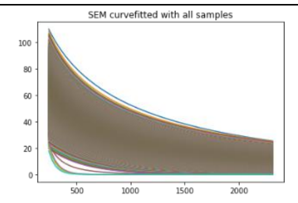
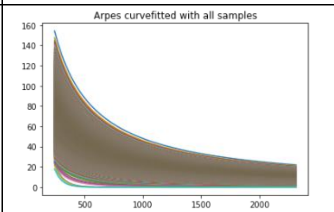
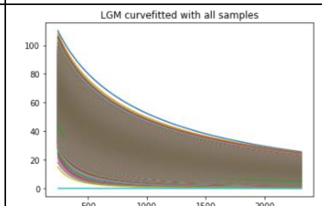
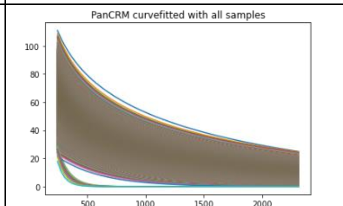
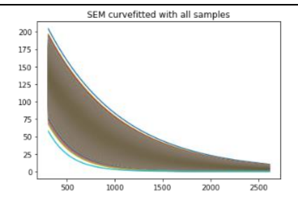
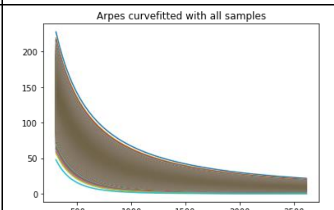
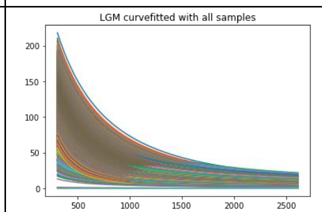
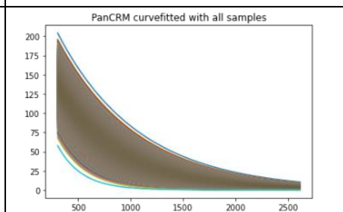
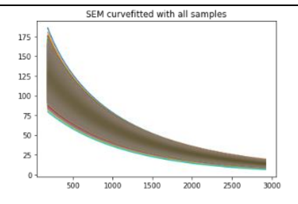
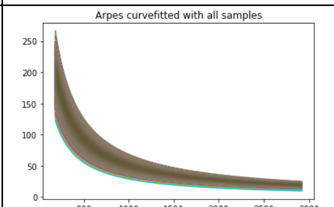
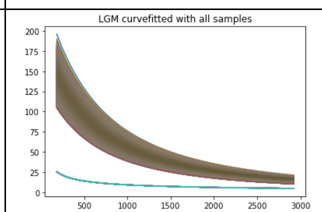
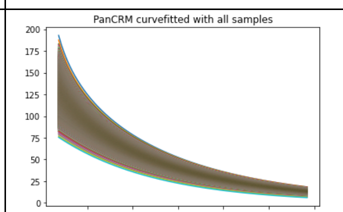
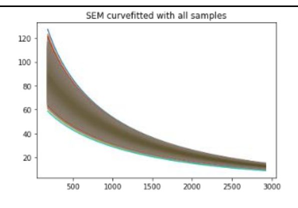
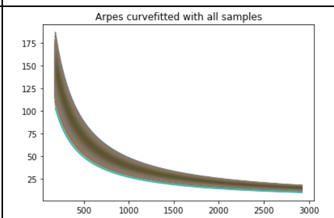
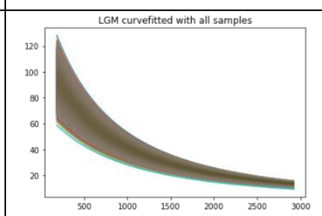
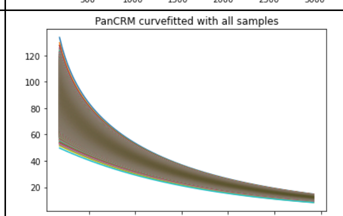
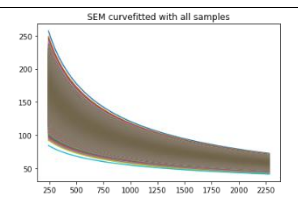
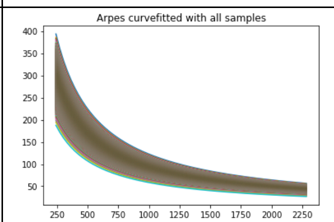
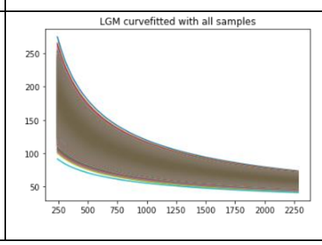
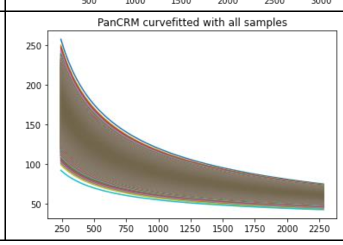
- Akbarnejad-Nesheli, B., Valko, P., Lee, J.W., 2012. Relating fracture Network char□acteristics to shale gas reserve estimation. In: SPE Americas Unconventional Resources Conference. Society of Petroleum Engineers. https://doi.org/10.2118/ 154841-MS

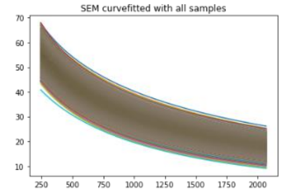
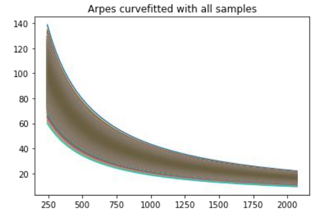
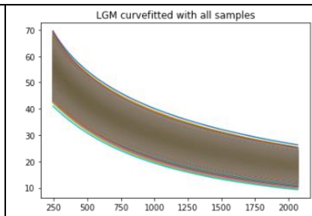
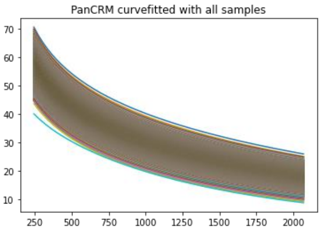
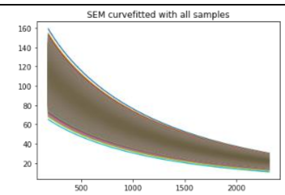
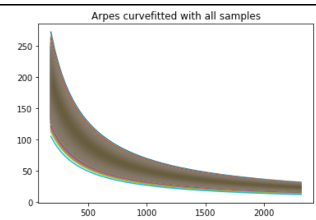
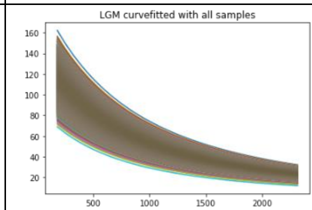
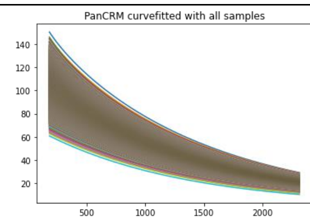
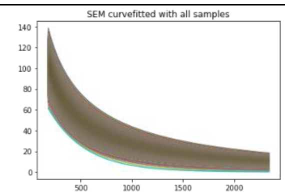
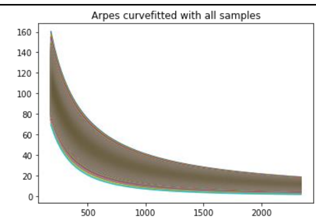
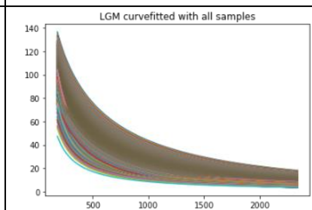
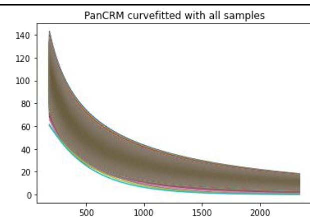
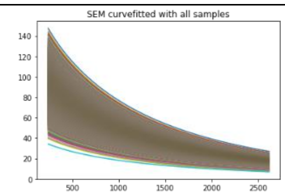
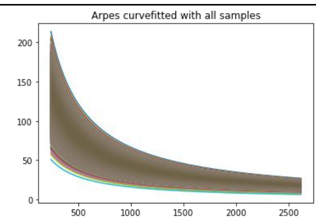
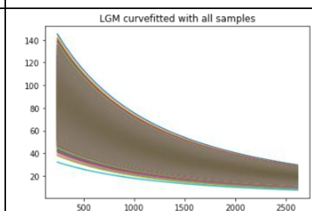
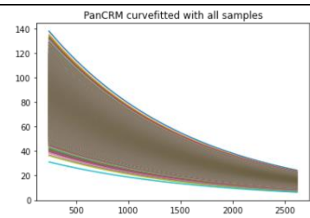
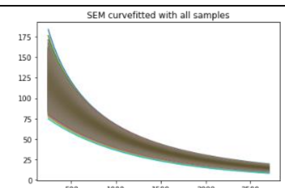
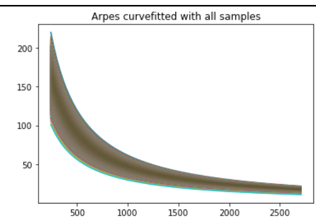
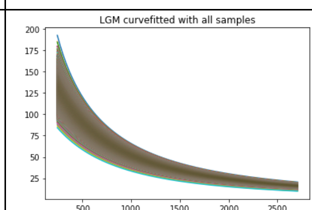
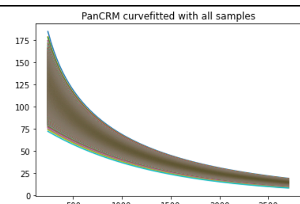
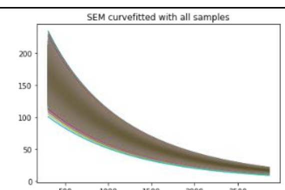
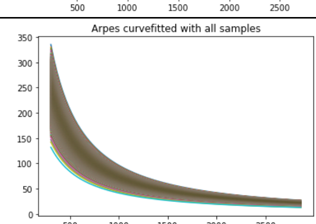
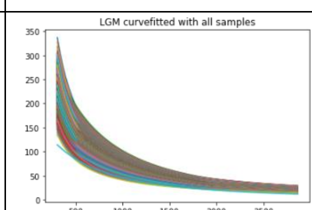
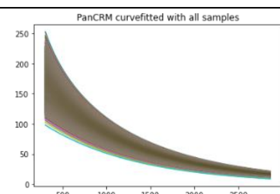
- Jouini, M. N., & Clemen, R. T. (1996). Copula models for aggregating expert opinions. *Operations Research*, *44*(3), 444-457.

Appendix A

Simulation graphs from the four models visualized for 10,000 samples

- All y-axis values are oil production B/d
- All x-axis values are time (Days)

•	SEM	Arpes	LGM	Pan CRM
9				
11				
13				
18				
19				
22				

24				
25				
32				
35				
40				
41				

Appendix B

Simulation code

In [1]:
```python
import random
import os
import string
import numpy as np
import matplotlib as plt
import matplotlib.pyplot as plt
import pandas as pd
import math
from scipy.optimize import curve_fit
from statistics import mean
from inspect import signature
```

In [ ]:
```python

```

CSV Extraction

--

In [2]:
```python
path = "C:/Users/maaz2/Documents/UiS/Thesis/Programs/Midland_AojieSelectedData.xlsx"
df0 = pd.io.excel.read_excel(path, sheet_name=2, header = 3)
```

In [3]:
```python
df0.drop('Unnamed: 0',inplace=True, axis=1)
```

In [4]:
```python
df0
```

Out[4]:

	Selected	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8	Unnamed: 9	Un
0	Well ID	8	9	11	13	14	15	16	17	

1 rows × 34 columns

In [5]:
```python
# Array of all the well numbers

wells = df0.iloc[0, 1:33]
```

In [6]:
```python
pdsheet = pd.io.excel.read_excel(path, sheet_name=0)
```

In [7]:
```python
pdsheet
```

Out[7]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8
0	NaN	NaN	Well ID	41	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	Date	dt	t	t_mid	Oil Rate [bbl/mon]	Oil Rate [bbl/day]	NaN
3	NaN	NaN	01/1965	31	31	15.5	0	0	NaN
4	NaN	NaN	02/1965	28	59	45	0	0	NaN

...	...	...	...	...	...	...	...	...	...
625	NaN	NaN	11/2016	30	18962	18947	591	19.7	NaN
626	NaN	NaN	12/2016	31	18993	18977.5	508	16.387097	NaN
627	NaN	NaN	01/2017	31	19024	19008.5	613	19.774194	NaN
628	NaN	NaN	02/2017	28	19052	19038	435	15.535714	NaN
629	NaN	NaN	03/2017	31	19083	19067.5	528	17.032258	NaN

630 rows × 12 columns

In [8]:

```
pdsheet.iloc[0:1, 3:4]
```

Out[8]:

	Unnamed: 3
0	41

In [9]:

```
pdsheet
```

Out[9]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8
0	NaN	NaN	Well ID	41	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	Date	dt	t	t_mid	Oil Rate [bbl/mon]	Oil Rate [bbl/day]	NaN
3	NaN	NaN	01/1965	31	31	15.5	0	0	NaN
4	NaN	NaN	02/1965	28	59	45	0	0	NaN
...	...	...	...	...	...	...	...	...	...
625	NaN	NaN	11/2016	30	18962	18947	591	19.7	NaN
626	NaN	NaN	12/2016	31	18993	18977.5	508	16.387097	NaN
627	NaN	NaN	01/2017	31	19024	19008.5	613	19.774194	NaN
628	NaN	NaN	02/2017	28	19052	19038	435	15.535714	NaN
629	NaN	NaN	03/2017	31	19083	19067.5	528	17.032258	NaN

630 rows × 12 columns

CSV Reading and cleaning and Plotting (WORKING)

In [10]:

```
# Reading file and making it into a dataframe

path = "C:/Users/maaz2/Documents/UiS/Thesis/Programs/Midland_AojieSelectedData.xlsx"
df1 = pd.io.excel.read_excel(path, sheet_name=0)
```

In [11]:

```
df1
```

Out[11]:

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	Unnamed: 6	Unnamed: 7	Unnamed: 8
0	NaN	NaN	Well ID	41	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN		NaN	NaN	NaN	NaN	NaN

2	NaN	NaN	Date	dt	t	t_mid	Oil Rate [bbl/mon]	Oil Rate [bbl/day]	NaN
3	NaN	NaN	01/1965	31	31	15.5	0	0	NaN
4	NaN	NaN	02/1965	28	59	45	0	0	NaN
...	...	...	...	...	...	...	...	...	...
625	NaN	NaN	11/2016	30	18962	18947	591	19.7	NaN
626	NaN	NaN	12/2016	31	18993	18977.5	508	16.387097	NaN
627	NaN	NaN	01/2017	31	19024	19008.5	613	19.774194	NaN
628	NaN	NaN	02/2017	28	19052	19038	435	15.535714	NaN
629	NaN	NaN	03/2017	31	19083	19067.5	528	17.032258	NaN

630 rows × 12 columns

In [12]:

```
#df.drop('Unnamed: 0',inplace=True, axis=1)
#df.drop('Unnamed: 1',inplace=True, axis=1)
```

In [13]:

```
df1 = pd.io.excel.read_excel(path, sheet_name=0, header = 3)
```

In [14]:

```
df1
```

Out[14]:

	Unnamed: 0	Unnamed: 1	Date	dt	t	t_mid	Oil Rate [bbl/mon]	Oil Rate [bbl/day]	Unnamed: 8	t.1	t_mid.1
0	NaN	NaN	01/1965	31	31	15.5	0	0.000000	NaN	31	15.5
1	NaN	NaN	02/1965	28	59	45.0	0	0.000000	NaN	59	45.0
2	NaN	NaN	03/1965	31	90	74.5	0	0.000000	NaN	90	74.5
3	NaN	NaN	04/1965	30	120	105.0	0	0.000000	NaN	120	105.0
4	NaN	NaN	05/1965	31	151	135.5	0	0.000000	NaN	151	135.5
...	...	...	...	...	...	...	...	...	...	...	...
622	NaN	NaN	11/2016	30	18962	18947.0	591	19.700000	NaN	18962	18947.0
623	NaN	NaN	12/2016	31	18993	18977.5	508	16.387097	NaN	18993	18977.5
624	NaN	NaN	01/2017	31	19024	19008.5	613	19.774194	NaN	19024	19008.5
625	NaN	NaN	02/2017	28	19052	19038.0	435	15.535714	NaN	19052	19038.0
626	NaN	NaN	03/2017	31	19083	19067.5	528	17.032258	NaN	19083	19067.5

627 rows × 12 columns

In [15]:

```
# Dropping the first 10 Columns as they are not needed

df1.drop(df1.iloc[:, 0:10], inplace = True, axis = 1)
```

In [16]:

```
df1
```

Out[16]:

	t_mid.1	Oil Rate [bbl/day].1
0	15.5	0.000000
1	45.0	0.000000
2	74.5	0.000000
3	105.0	0.000000

4	135.5	0.000000
...	...	...
622	18947.0	19.700000
623	18977.5	16.387097
624	19008.5	19.774194
625	19038.0	15.535714
626	19067.5	17.032258

627 rows × 2 columns

In [17]:

```
#Checking
df1['Oil Rate [bbl/day].1']
```

Out[17]:

```
0       0.000000
1       0.000000
2       0.000000
3       0.000000
4       0.000000
          ...
622    19.700000
623    16.387097
624    19.774194
625    15.535714
626    17.032258
Name: Oil Rate [bbl/day].1, Length: 627, dtype: float64
```

In [18]:

```
# Dropping all zero rows

df1 = df1.loc[df1['Oil Rate [bbl/day].1'] != 0]
```

In [19]:

```
df1
```

Out[19]:

	t_mid.1	Oil Rate [bbl/day].1
532	16206.5	49.548387
533	16237.0	313.100000
534	16267.5	324.096774
535	16298.5	291.967742
536	16329.0	253.866667
...	...	...
622	18947.0	19.700000
623	18977.5	16.387097
624	19008.5	19.774194
625	19038.0	15.535714
626	19067.5	17.032258

95 rows × 2 columns

In [20]:

```
# Indexing the dataframe so it starts from zero
df1 = df1.reset_index(drop=True)
```

In [21]:

```
df1
```

Out[21]:

	t_mid.1	Oil Rate [bbl/day].1
0	16206.5	49.548387
1	16237.0	313.100000
2	16267.5	324.096774
3	16298.5	291.967742
4	16329.0	253.866667
...	...	...
90	18947.0	19.700000
91	18977.5	16.387097
92	19008.5	19.774194
93	19038.0	15.535714
94	19067.5	17.032258

95 rows × 2 columns

In [22]:

```
# Selecting date of extraction start so that we can set time = 0

Extraction_Start_Datedf = df1.iloc[0,0]
```

In [23]:

```
df1['t_mid.1'] = df1['t_mid.1'] - Extraction_Start_Datedf
```

In [24]:

```
df1
```

Out[24]:

	t_mid.1	Oil Rate [bbl/day].1
0	0.0	49.548387
1	30.5	313.100000
2	61.0	324.096774
3	92.0	291.967742
4	122.5	253.866667
...	...	...
90	2740.5	19.700000
91	2771.0	16.387097
92	2802.0	19.774194
93	2831.5	15.535714
94	2861.0	17.032258

95 rows × 2 columns

In [25]:

```
# Finding maximum so we can use that as a starting point

max(df1['Oil Rate [bbl/day].1'])
```

Out[25]:

```
324.0967741935484
```

In [26]:

```
df1.loc[df1['Oil Rate [bbl/day].1'] == max(df1['Oil Rate [bbl/day].1'])]
```

Out[26]:

	t_mid.1	Oil Rate [bbl/day].1
2	61.0	324.096774

In [27]:

```
#Dropping values before maximum

index_value = df1[df1['Oil Rate [bbl/day].1']== max(df1['Oil Rate [bbl/day].1'])].inc
```

In [28]:

```
index_value[0]
```

Out[28]: 2

In [29]:

```
#Dropping values before maximum
df1 = df1.drop(df1.index[0])
df1 = df1.drop(df1.index[:index_value[0]])
```

In [30]:

```
df1
```

Out[30]:

	t_mid.1	Oil Rate [bbl/day].1
3	92.0	291.967742
4	122.5	253.866667
5	153.0	255.000000
6	183.5	255.000000
7	214.0	242.548387
...	...	...
90	2740.5	19.700000
91	2771.0	16.387097
92	2802.0	19.774194
93	2831.5	15.535714
94	2861.0	17.032258

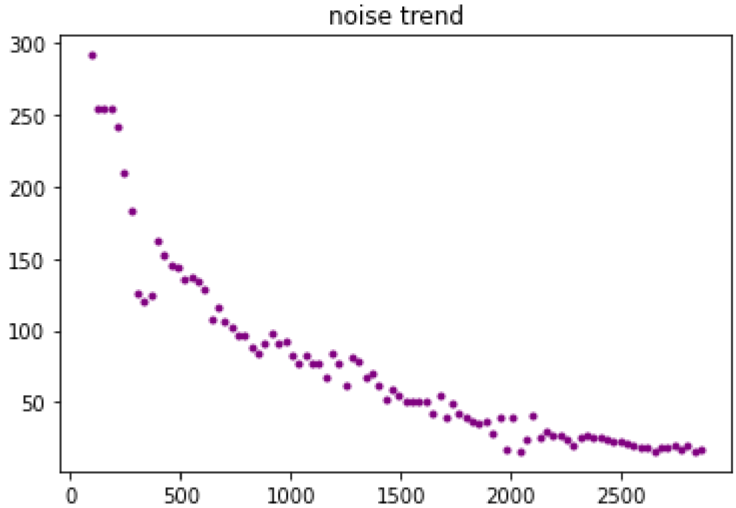
92 rows × 2 columns

In [31]:

```
plt.plot(df1['t_mid.1'], df1['Oil Rate [bbl/day].1'], '.', color = 'purple')
plt.title('noise trend')
```

Out[31]:

```
Text(0.5, 1.0, 'noise trend')
```



In [32]:
```
# assigning variables
t = df1['t_mid.1']
qt_setup = df1['Oil Rate [bbl/day].1']
```

Implementing SEM Model with Curvefit

In [33]:
```
def sem(days,q0, n, tau):
    qt = q0 * np.exp(-(days/tau)**n)
    return qt
```

In [34]:
```
par_sem, cov_sem = curve_fit(sem, t, qt_setup, p0 = [max(df1['Oil Rate [bbl/day].1'])
```

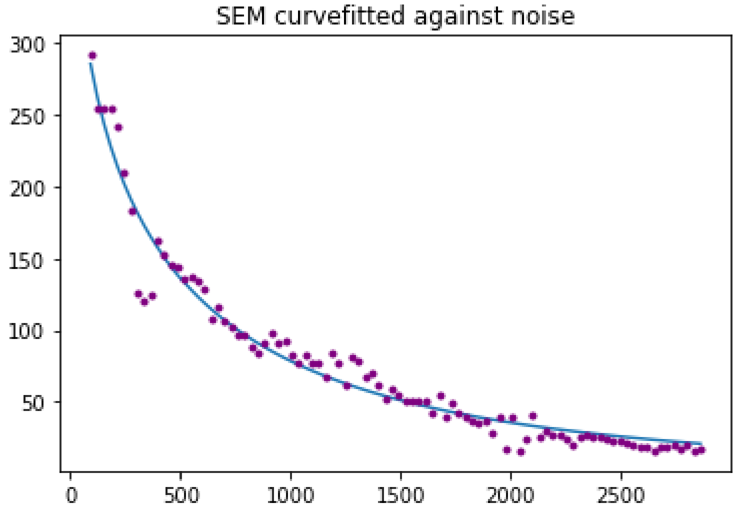
In [35]:
```
# _cf referres to curve fitted as we are using the parameters to make a new decline c

q0_sem_cf = par_sem[0]
n_sem_cf = par_sem[1]
tau_sem_cf = par_sem[2]
```

In [36]:
```
par_sem
```

Out[36]:
```
array([478.31134486,   0.52630254, 324.2406865 ])
```

In [37]:
```
qt_sem_curve_fitted = sem(t,q0_sem_cf,n_sem_cf,tau_sem_cf)
```

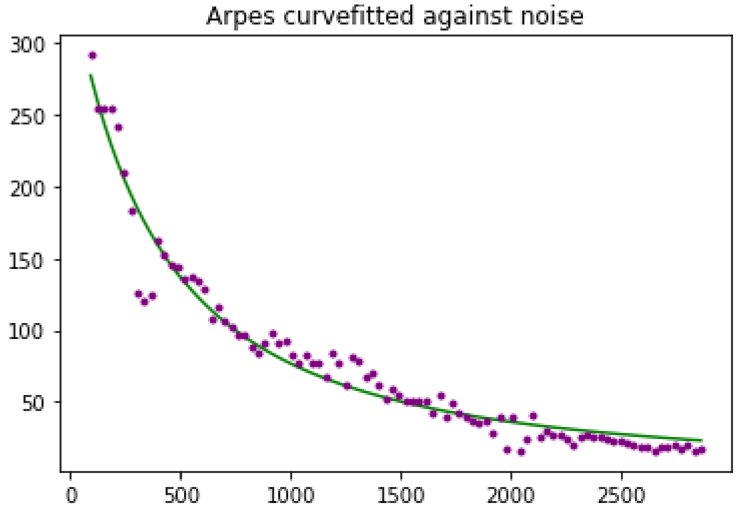
In [38]:
```
plt.title('SEM curvefitted against noise')
plt.plot(t, qt_sem_curve_fitted)
sem_plot = plt.plot(df1['t_mid.1'], df1['Oil Rate [bbl/day].1'], '.', color = 'purple
```



In [39]:
```
sem_plot
```
Out[39]:
```
[<matplotlib.lines.Line2D at 0x184e1062910>]
```

Implementing Arpes with Curvefit

In [40]:
```
def arpes(days, q0_arpes, b, Di):
    qt_Arpes = q0_arpes *(1 + b*Di*days)**(-1/b)
    return qt_Arpes
```
In [41]:
```
par_arpes, cov_arpes = curve_fit(arpes, t, qt_setup, bounds=((0, 0, 0), (np.inf, 1, r
```
In [42]:
```
par_arpes
```
Out[42]:
```
array([3.44212386e+02, 6.27271607e-01, 2.50055692e-03])
```
In [43]:
```
# _cf referres to curve fitted as we are using the parameters to make a new decline c

q0_arpes_cf = par_arpes[0]
b_arpes_cf = par_arpes[1]
Di_arpes_cf = par_arpes[2]
```
In [44]:
```
qt_arpes_curve_fitted = arpes(t,q0_arpes_cf, b_arpes_cf, Di_arpes_cf)
```
In [45]:
```
plt.title('Arpes curvefitted against noise')
plt.plot(t, qt_arpes_curve_fitted, color = "green")
plt.plot(df1['t_mid.1'], df1['Oil Rate [bbl/day].1'], '.', color = 'purple')
```
Out[45]:
```
[<matplotlib.lines.Line2D at 0x184e10d3f10>]
```



Implementing LGM with Curvefit

In [46]:
```
a_0 = np.random.randint(10, df1['t_mid.1']. iloc[-1])
eta_0 = np.random.randint(1, 100)/100
k_0 = max(df1['Oil Rate [bbl/day].1'])*a_0/eta_0
```

In [47]:
```
def lgm(days, a, k, eta):
    qt_LGM = a*k*eta*days**(eta - 1)  / (a + days**eta)**2
    return qt_LGM
```

In [48]:
```
par_lgm, cov_lgm = curve_fit(lgm, t, qt_setup, p0 = [a_0,  k_0 ,eta_0], bounds=((0.00
```

In [49]:
```
# Values of LGM = [100, 209, 0.99] vs [113.5, 206, 0.87]
par_lgm
```

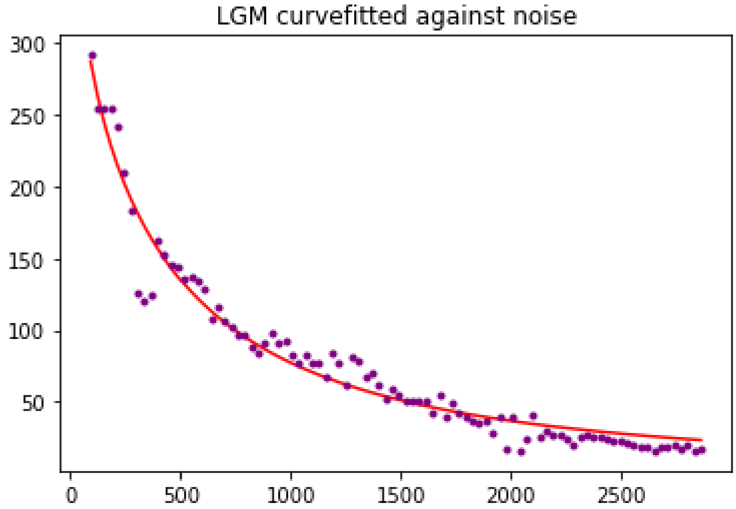
Out[49]:
```
array([5.47407120e+02, 3.46667519e+05, 8.93837200e-01])
```

In [50]:
```
a_lgm_cf = par_lgm[0]
k_lgm_cf = par_lgm[1]
eta_lgm_cf = par_lgm[2]
```

In [51]:
```
qt_lgm_curve_fitted = lgm(t, a_lgm_cf, k_lgm_cf ,eta_lgm_cf )
```

In [52]:
```
plt.title('LGM curvefitted against noise')
plt.plot(t, qt_lgm_curve_fitted, color = 'red')
plt.plot(df1['t_mid.1'], df1['Oil Rate [bbl/day].1'], '.', color = 'purple')
```

Out[52]:
```
[<matplotlib.lines.Line2D at 0x184e1149640>]
```



Implementing Pan CRM with Curvefit

In [53]:
```
def panCRM(t, delta_P, J_inf, ctVp, Beta):

    J = Beta/(np.sqrt(t)) + J_inf

    #qt_Pan_CRM = delta_P*J*np.exp(-(2*Beta*np.sqrt(t) + J_inf*t)/ctVp)
    qt_Pan_CRM = delta_P*J*np.exp((-2*J*t)/ctVp)

    return qt_Pan_CRM
```

In [54]:
```
# p0 Parameters need to be better. we were missing one parameter i.e. 3 instead of 4


#par_panCRM, cov_panCRM = curve_fit(panCRM, t, qt_setup)
par_panCRM, cov_panCRM = curve_fit(panCRM, t, qt_setup, p0 = [1000, 1, 200, 2], bounc
```

In [55]:
```
par_panCRM
```

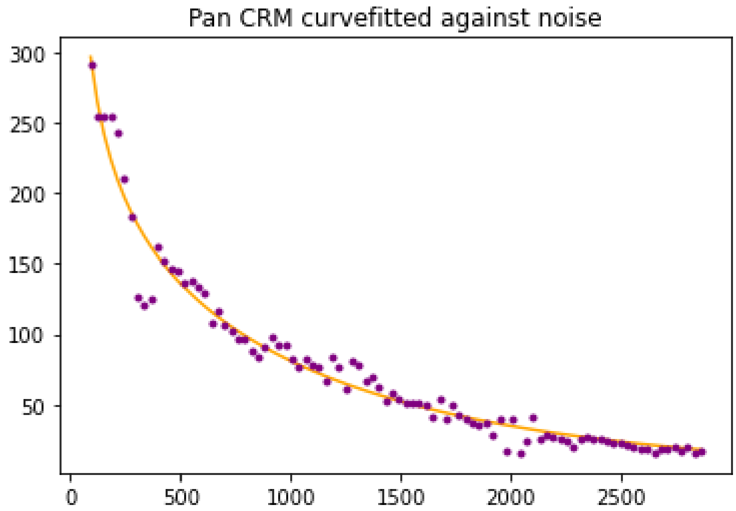
Out[55]:
```
array([5.39277417e+02, 2.61402554e-01, 8.18387459e+02, 3.57829784e+00])
```

In [56]:
```
delta_P_panCRM_cf = par_panCRM[0]
J_inf_panCRM_cf = par_panCRM[1]
ctVp_panCRM_cf = par_panCRM[2]
Beta_panCRM_cf = par_panCRM[3]
```

In [57]:
```
qt_panCRM_curve_fitted = panCRM(t, delta_P_panCRM_cf, J_inf_panCRM_cf, ctVp_panCRM_cf
```

In [58]:
```
plt.title('Pan CRM curvefitted against noise')
plt.plot(t, qt_panCRM_curve_fitted, color = 'orange')
plt.plot(df1['t_mid.1'], df1['Oil Rate [bbl/day].1'], '.', color = 'purple')
```

Out[58]:
```
[<matplotlib.lines.Line2D at 0x184e11b8700>]
```



Moving Window (Pandas Version)

In [59]:
```
df = pd.DataFrame({'Time':df1['t_mid.1'], 'Data':df1['Oil Rate [bbl/day].1'], 'SEM':
```

In [60]:
```
df
```

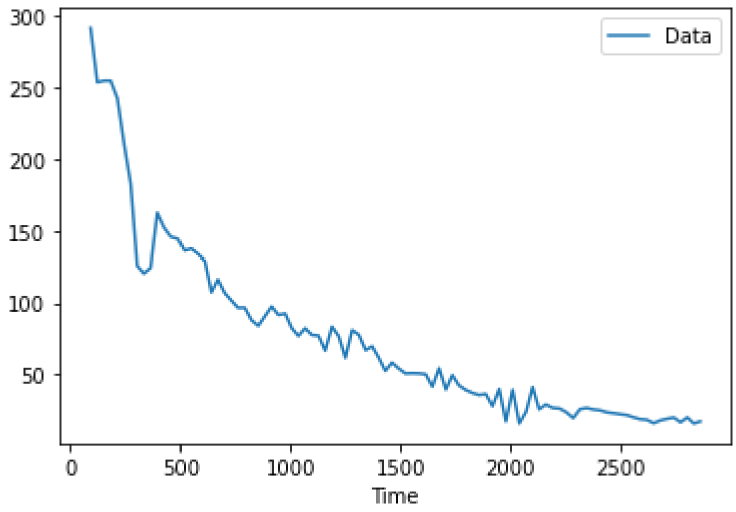
Out[60]:

	Time	Data	SEM	Arpes	LGM	Pan CRM
3	92.0	291.967742	285.702053	277.650886	287.374863	296.666883
4	122.5	253.866667	262.733371	260.101222	264.062645	264.684632
5	153.0	255.000000	243.902723	244.287684	244.981736	241.710738
6	183.5	255.000000	227.956298	229.981072	228.793010	223.911681
7	214.0	242.548387	214.152049	216.989494	214.741063	209.431827
...	...	...	...	...	...	...
90	2740.5	19.700000	22.090287	24.119646	24.456039	19.537717
91	2771.0	16.387097	21.696977	23.776492	24.098646	19.090249
92	2802.0	19.774194	21.306435	23.435780	23.743656	18.646547
93	2831.5	15.535714	20.943167	23.118869	23.413343	18.234431
94	2861.0	17.032258	20.587835	22.808874	23.090125	17.831926

92 rows × 6 columns

In [61]:
```
df.plot.line(x='Time', y='Data')
```

Out[61]:
```
<AxesSubplot:xlabel='Time'>
```



In [62]:

```
df['MovingAverage'] = df['Data'].rolling(5).mean()
df['MovingSD'] = df['Data'].rolling(5).std()
```

In [63]:

```
df
```

Out[63]:

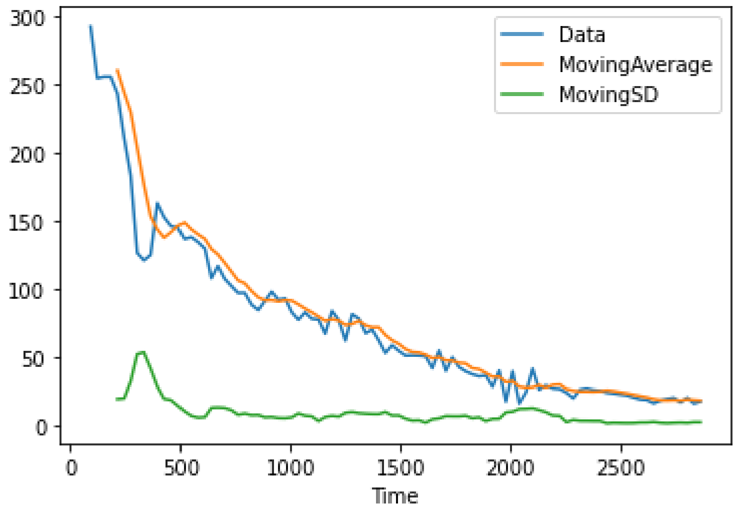
	Time	Data	SEM	Arpes	LGM	Pan CRM	MovingAverage	MovingSD
3	92.0	291.967742	285.702053	277.650886	287.374863	296.666883	NaN	NaN
4	122.5	253.866667	262.733371	260.101222	264.062645	264.684632	NaN	NaN
5	153.0	255.000000	243.902723	244.287684	244.981736	241.710738	NaN	NaN
6	183.5	255.000000	227.956298	229.981072	228.793010	223.911681	NaN	NaN
7	214.0	242.548387	214.152049	216.989494	214.741063	209.431827	259.676559	18.798870
...	...	...	...	...	...	...	...	...
90	2740.5	19.700000	22.090287	24.119646	24.456039	19.537717	18.028602	1.481212
91	2771.0	16.387097	21.696977	23.776492	24.098646	19.090249	17.680215	1.647236
92	2802.0	19.774194	21.306435	23.435780	23.743656	18.646547	18.486667	1.434359
93	2831.5	15.535714	20.943167	23.118869	23.413343	18.234431	18.047143	1.962412
94	2861.0	17.032258	20.587835	22.808874	23.090125	17.831926	17.685853	1.946471

92 rows × 8 columns

In [64]:

```
df.plot.line(x='Time', y=['Data', 'MovingAverage', 'MovingSD'])
```

Out[64]: <AxesSubplot:xlabel='Time'>



Ranking

In [65]:
```
# Converting Data Frame to array

Moving_avg = df.MovingAverage
Moving_SD = df.MovingSD
```

In [66]:
```
distributed = np.random.normal(loc= Moving_avg, scale = Moving_SD)
```

In [67]:
```
# Number of samples can be defined here for scaling

samples = []
counter = 0
Number_of_Samples = 10000
```

In [68]:
```
while counter < Number_of_Samples:
    sample_distributed = np.random.normal(loc= Moving_avg, scale = Moving_SD)
    samples.append(sample_distributed)
    counter = counter + 1
```

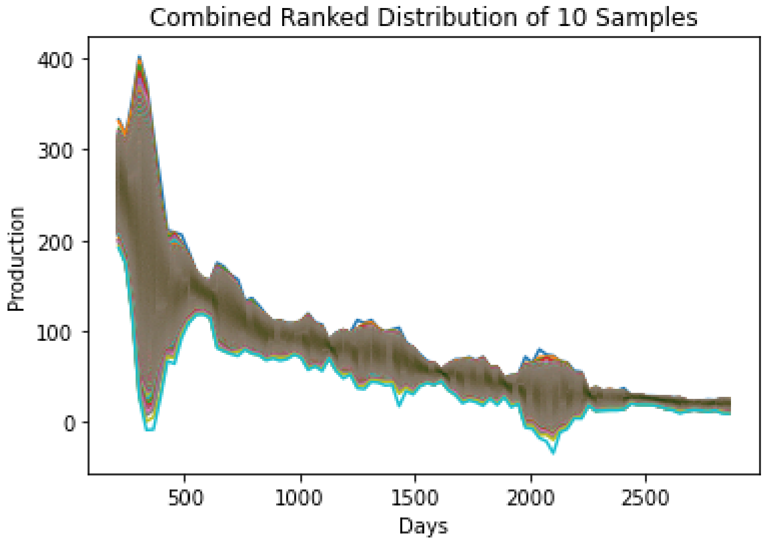
In [70]:
```
# Converting to NumPy array
samples = np.asarray(samples)
```

In [71]:
```
# Checking shape (samples, DataPoints)
samples.shape
```

Out[71]:
```
(10000, 92)
```

In [72]:
```
# Array is sorted by column in decending

sorted_rank = -np.sort(-samples, axis=0)
```

In [73]:
```
# Combined representation of all the ranked data points and their trends

plt.title('Combined Ranked Distribution of 10 Samples')
plt.xlabel('Days')
plt.ylabel('Production')
f =0
array_sorted_rank = []
while f < len(samples):
    plt.plot(t, sorted_rank[f], label = f)
    array_sorted_rank.append(sorted_rank[f])
    f= f+1
#plt.legend()
```



In [74]:

```
# Checking Only
np.shape(array_sorted_rank)
```

Out[74]:

```
(10000, 92)
```

In [75]:

```
# Checking Only
len(array_sorted_rank)
```

Out[75]:

```
10000
```

In [76]:

```
counter_samples = 0
label_samples = []
while counter_samples < len(array_sorted_rank):
    label_samples.append(str(counter_samples))
    counter_samples = counter_samples +1
```

In [77]:

```
df3 = pd.DataFrame(np.transpose(array_sorted_rank), columns =label_samples)
df3['t'] = t
```

In [78]:

```
# Appending SD to DF 3

df3["MovingSD"] = Moving_SD
```

In [79]:

```
df3
```

Out[79]:

	0	1	2	3	4	5	6	7	
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	332.129455	329.870626	321.833373	319.136906	318.479942	317.699318	317.208695	316.800002	316.2
...	...	...	...	...	...	...	...	...	
87	23.276881	22.889304	22.868435	22.858592	22.821990	22.789532	22.780610	22.664896	22.6
88	24.266867	23.636839	23.634897	23.263044	23.041262	23.009386	22.966927	22.949025	22.8
89	23.729897	23.559114	23.539558	23.495218	23.313103	23.231697	23.163170	23.136922	23.0
90	25.484315	24.609905	24.568558	24.340222	24.319740	24.304685	24.245795	24.196162	24.1

91	25.387970	24.781493	24.549008	24.118512	23.937950	23.932595	23.874951	23.857205	23.8

92 rows × 10002 columns

In [80]:

```
# Removing Nan

df3 = df3.iloc[4: , :]
```

In [81]:

```
df3
```

Out[81]:

	0	1	2	3	4	5	6	7	
4	332.129455	329.870626	321.833373	319.136906	318.479942	317.699318	317.208695	316.800002	316.2
5	315.892277	315.029960	307.653855	307.455687	306.241817	305.646216	305.166078	304.981206	304.0
6	355.159653	347.003035	336.214698	333.443283	331.399598	329.959165	329.642725	328.756351	328.4
7	401.330452	397.461744	391.537208	384.676592	377.138674	369.849097	369.528463	368.258964	366.5
8	372.664780	366.104397	363.815535	361.359333	356.521256	356.008060	355.616623	350.294883	345.7
...	...	...	...	...	...	...	...	...	
87	23.276881	22.889304	22.868435	22.858592	22.821990	22.789532	22.780610	22.664896	22.6
88	24.266867	23.636839	23.634897	23.263044	23.041262	23.009386	22.966927	22.949025	22.8
89	23.729897	23.559114	23.539558	23.495218	23.313103	23.231697	23.163170	23.136922	23.0
90	25.484315	24.609905	24.568558	24.340222	24.319740	24.304685	24.245795	24.196162	24.1
91	25.387970	24.781493	24.549008	24.118512	23.937950	23.932595	23.874951	23.857205	23.8

88 rows × 10002 columns

In [82]:

```
# Dropping all non values

df3 = df3.dropna()
```

In [83]:

```
df3
```

Out[83]:

	0	1	2	3	4	5	6	7	
7	401.330452	397.461744	391.537208	384.676592	377.138674	369.849097	369.528463	368.258964	366.5
8	372.664780	366.104397	363.815535	361.359333	356.521256	356.008060	355.616623	350.294883	345.7
9	316.366666	307.830207	305.697717	298.436967	292.435951	290.640131	287.196910	287.179428	286.8
10	264.754502	245.844393	242.788581	237.778934	233.191976	231.577757	231.355198	229.022037	228.7
11	210.691445	206.251365	203.317346	197.899284	197.225987	195.553613	195.265564	194.809351	193.6
...	...	...	...	...	...	...	...	...	
87	23.276881	22.889304	22.868435	22.858592	22.821990	22.789532	22.780610	22.664896	22.6
88	24.266867	23.636839	23.634897	23.263044	23.041262	23.009386	22.966927	22.949025	22.8
89	23.729897	23.559114	23.539558	23.495218	23.313103	23.231697	23.163170	23.136922	23.0
90	25.484315	24.609905	24.568558	24.340222	24.319740	24.304685	24.245795	24.196162	24.1
91	25.387970	24.781493	24.549008	24.118512	23.937950	23.932595	23.874951	23.857205	23.8

85 rows × 10002 columns

In [84]:

```
# Len is Samples+2 due to the time and SD columns so we need to remove it

len(df3.columns)
```

Out[84]:
```
10002
```

In [85]:
```
# Important to set the values to be appended

col_name = df3.columns[0:len(df3.columns) -2]
```

In [86]:
```
# Checking if values of colums are correct

col_name
```

Out[86]:
```
Index(['0', '1', '2', '3', '4', '5', '6', '7', '8', '9',
       ...
       '9990', '9991', '9992', '9993', '9994', '9995', '9996', '9997', '9998',
       '9999'],
      dtype='object', length=10000)
```

In [87]:
```
# To collect sample curvefit
sem_sample_array = []
```

In [88]:
```
df3["MovingSD"]
```

Out[88]:
```
7     18.798870
8     19.262478
9     31.726042
10    51.651471
11    52.996862
        ...
87     2.117634
88     1.497629
89     1.221159
90     1.481212
91     1.647236
Name: MovingSD, Length: 85, dtype: float64
```

In [89]:
```
df1 = df1.tail(len(df3))
```

In [90]:
```
t = t.tail(len(df3))
```

In [91]:
```
t
```

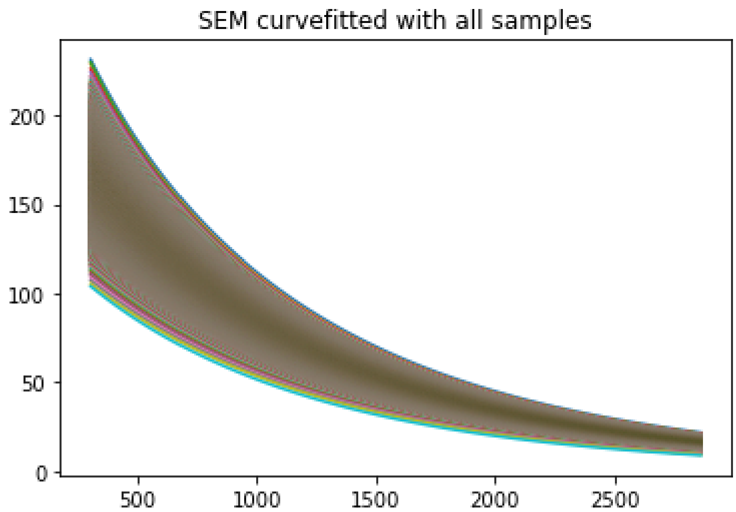
Out[91]:
```
10     304.0
11     334.5
12     365.0
13     395.5
14     426.0
       ...
90    2740.5
91    2771.0
92    2802.0
93    2831.5
94    2861.0
Name: t_mid.1, Length: 85, dtype: float64
```

Implementiong SEM

In [ ]:

In [92]:

```
# To collect sample curvefit
sem_sample_array = []

# Automated Curvefit for all samples of SEM


# starting point p0 = [max(Noise), 30, 1]

for x in col_name:
    sample_col = np.asarray(df3[x])
    time_converted = np.asarray(df3.t)
    par_sem, cov_sem = curve_fit(sem, time_converted, sample_col, sigma=df3["MovingSD
    q0_sem_cf = par_sem[0]
    n_sem_cf = par_sem[1]
    tau_sem_cf = par_sem[2]


    qt_sem_curve_fitted = sem(t,q0_sem_cf,n_sem_cf,tau_sem_cf)
    sem_sample_array.append(qt_sem_curve_fitted)  # Changed from 5 to 6 for well 24||

    plt.title('SEM curvefitted with all samples')
    plt.plot(t, qt_sem_curve_fitted)
```



In [93]:

```
np.shape(sem_sample_array)
```

Out[93]:

```
(10000, 85)
```

In [94]:

```
# Confirming shape. Should be (samples, DataPoints)

np.shape(sem_sample_array)
```

Out[94]:

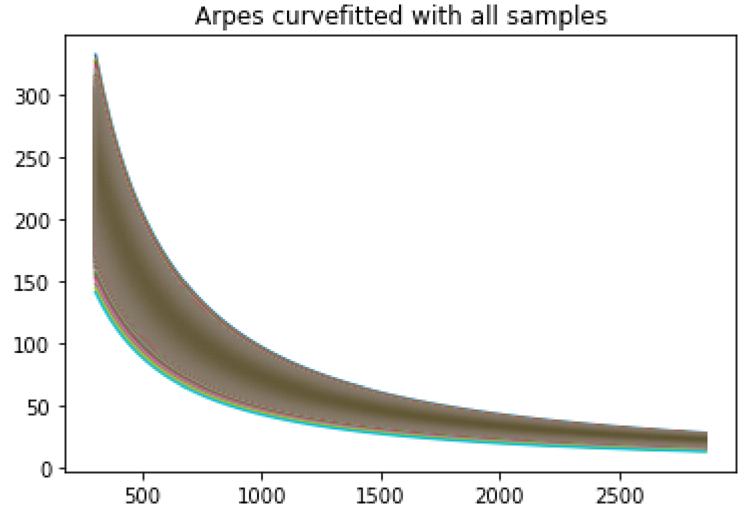
```
(10000, 85)
```

Implementing Curvefit for Arpes

In [95]:

```
# To collect sample curvefit
arpes_sample_array = []

# Automated Curvefit for all samples of Arpes
```

```
# p0 [max(Noise), 0.5, 0.05]

for x in col_name:
    sample_col = np.asarray(df3[x])
    time_converted = np.asarray(df3.t)
    par_arpes, cov_arpes = curve_fit(arpes, time_converted, sample_col, sigma=df3["Mo
    q0_arpes_cf = par_arpes[0]
    b_arpes_cf = par_arpes[1]
    Di_arpes_cf = par_arpes[2]


    qt_arpes_curve_fitted = arpes(t,q0_arpes_cf,b_arpes_cf,Di_arpes_cf)
    arpes_sample_array.append(qt_arpes_curve_fitted) # Changed from 5 to 6 well 40

    plt.title('Arpes curvefitted with all samples')
    plt.plot(t, qt_arpes_curve_fitted)
```



In [96]:
```
np.shape(arpes_sample_array)
```

Out[96]:
```
(10000, 85)
```

Implementing Curvefit for LGM

In [97]:
```
3*max(t)
```

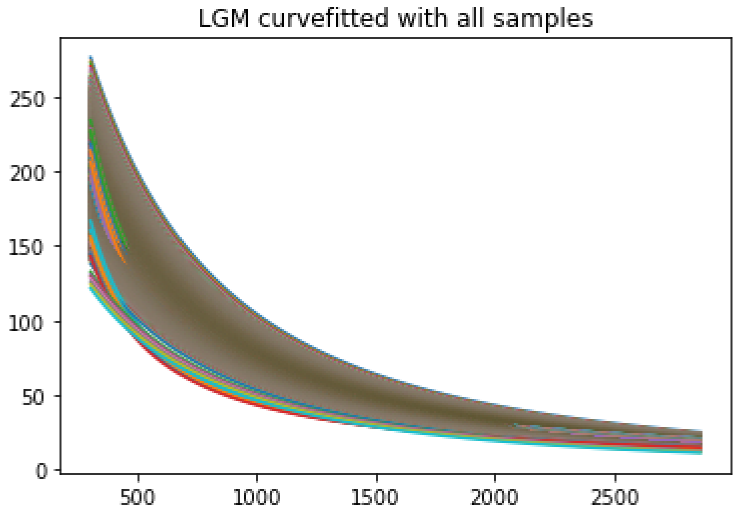
Out[97]:
```
8583.0
```

In [98]:
```
df3["0"]
```

Out[98]:
```
7     401.330452
8     372.664780
9     316.366666
10    264.754502
11    210.691445
         ...
87     23.276881
88     24.266867
89     23.729897
90     25.484315
91     25.387970
Name: 0, Length: 85, dtype: float64
```

In [99]:
```
a_0 = np.random.randint(10, 3*max(t))
eta_0 = np.random.randint(1, 100)/100
k_0 = max(df3["0"])*a_0/eta_0
```

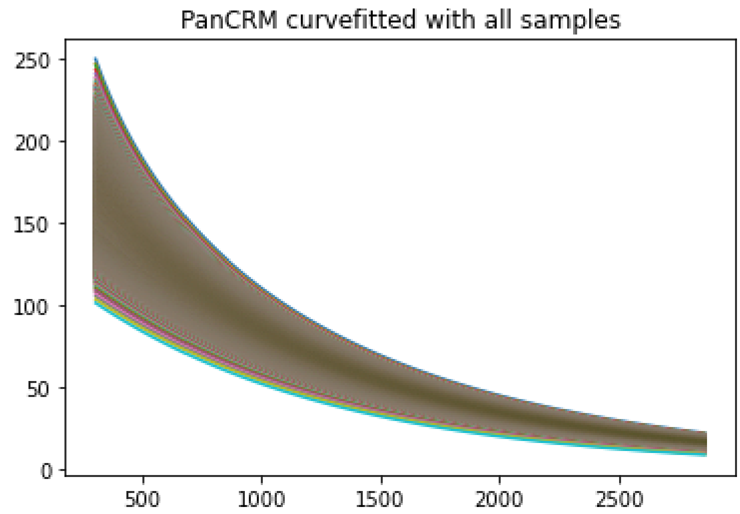
In [100…
```
# To collect sample curvefit
lgm_sample_array = []

# Automated Curvefit for all samples of SEM
# p0 [50, (max(Noise)*50)/0.5, 0.5 ]


for x in col_name:
    sample_col = np.asarray(df3[x])
    time_converted = np.asarray(df3.t)
    par_lgm, cov_lgm = curve_fit(lgm, time_converted, sample_col, p0 = [a_0, k_0, eta
    a_lgm_cf = par_lgm[0]
    k_lgm_cf = par_lgm[1]
    eta_lgm_cf = par_lgm[2]


    qt_lgm_curve_fitted = lgm(t,a_lgm_cf,k_lgm_cf,eta_lgm_cf)
    lgm_sample_array.append(qt_lgm_curve_fitted) # Changed from 5 to 6 well 40

    plt.title('LGM curvefitted with all samples')
    plt.plot(t, qt_lgm_curve_fitted)
```



Implementing Curvefit for Pan CRM

In [101…
```
# To collect sample curvefit
panCRM_sample_array = []

# Automated Curvefit for all samples of SEM
#p0 = [np.random.randint(500, 1500), np.random.randint(5, 15)*0.1, np.random.randint(


for x in col_name:
    sample_col = np.asarray(df3[x])
    time_converted = np.asarray(df3.t)
    par_panCRM, cov_panCRM = curve_fit(panCRM, time_converted, sample_col, p0 = [np.r
    delta_P_panCRM_cf = par_panCRM[0]
    J_inf_panCRM_cf = par_panCRM[1]
    ctVp_panCRM_cf = par_panCRM[2]
    Beta_panCRM_cf = par_panCRM[3]

    qt_PanCRM_curve_fitted = panCRM(t, delta_P_panCRM_cf, J_inf_panCRM_cf, ctVp_panCR
    panCRM_sample_array.append(qt_PanCRM_curve_fitted)
```

```
    plt.title('PanCRM curvefitted with all samples')
    plt.plot(t, qt_PanCRM_curve_fitted)
```



In [ ]:

```
```

In [102…

```
np.shape(panCRM_sample_array)
```

Out[102…

```
(10000, 85)
```

In [103…

```
len((panCRM_sample_array[0]))
```

Out[103…

```
85
```

Combined arrayes to set up maximum likihood

Maximum Liklihood Function (Not in use in new implementation)

In [104…

```
def MaximumLikelihood(arraya, arrayb, arrayc):
    i = 0
    np.d= []
    while i < len(arraya):
        c = (arraya[i] - arrayb[i])**2     / (arrayc[i])**2
        np.d.append(c)
        c
        i = i+1
    return np.sum(np.d)
```

SEM

In [105…

```
#df3["MovingSD"]
```

In [106…

```
len(np.array(sem_sample_array))
```

Out[106…
```
10000
```

In [107…
```
len(np.array(df1['Oil Rate [bbl/day].1'][6:]))
```

Out[107…
```
79
```

In [108…
```
len(np.array(df3["MovingSD"]))
```

Out[108…
```
85
```

In [109…
```
len(sem_sample_array[0])
```

Out[109…
```
85
```

In [110…
```
df1.tail(len(df3))
```

Out[110…

	t_mid.1	Oil Rate [bbl/day].1
10	304.0	125.903226
11	334.5	120.466667
12	365.0	124.419355
13	395.5	162.566667
14	426.0	152.225806
...	...	...
90	2740.5	19.700000
91	2771.0	16.387097
92	2802.0	19.774194
93	2831.5	15.535714
94	2861.0	17.032258

85 rows × 2 columns

In [111…
```
df1[:len(df3)]
```

Out[111…

	t_mid.1	Oil Rate [bbl/day].1
10	304.0	125.903226
11	334.5	120.466667
12	365.0	124.419355
13	395.5	162.566667
14	426.0	152.225806
...	...	...
90	2740.5	19.700000
91	2771.0	16.387097
92	2802.0	19.774194
93	2831.5	15.535714
94	2861.0	17.032258

85 rows × 2 columns

In [112…
```
len(np.array(df1['Oil Rate [bbl/day].1']))
```
Out[112…
```
85
```
In [113…
```
len(t)
```
Out[113…
```
85
```
In [114…
```
len(np.array(sem_sample_array[0]))
```
Out[114…
```
85
```
In [115…
```
len(np.array(df3["MovingSD"]))
```
Out[115…
```
85
```
In [116…
```
test_count = 0
ML_sem_test_array = []
while test_count< len(np.array(sem_sample_array)):
    Noise_Data_Test = np.array(df1['Oil Rate [bbl/day].1']) # Changed from 5 to 6 wei
    Model_Data_Test = np.array(sem_sample_array[test_count])
    Std_Dev_Test = np.array(df3["MovingSD"])

    sum_count_test = np.sum((Noise_Data_Test - Model_Data_Test)**2   / (Std_Dev_Test)
    test_count = test_count +1
    ML_sem_test_array.append(sum_count_test)
#print(ML_sem_test_array)
```

Arpes

In [117…
```
len(arpes_sample_array[0])
```
Out[117…
```
85
```
In [118…
```
len(np.array(df1['Oil Rate [bbl/day].1']))
```
Out[118…
```
85
```
In [119…
```
len(np.array(df3["MovingSD"]))
```
Out[119…
```
85
```
In [120…
```
test_count = 0
ML_arpes_test_array = []
while test_count< len(np.array(arpes_sample_array)):
    Noise_Data_Test = np.array(df1['Oil Rate [bbl/day].1']) # Changed from 5 to 6 wei
    Model_Data_Test = np.array(arpes_sample_array[test_count])
    Std_Dev_Test = np.array(df3["MovingSD"])

    sum_count_test = np.sum((Noise_Data_Test - Model_Data_Test)**2   / (Std_Dev_Test)
    test_count = test_count +1
```

```
    ML_arpes_test_array.append(sum_count_test)
#print(ML_arpes_test_array)
```

LGM

In [121…

```
test_count = 0
ML_lgm_test_array = []
while test_count< len(np.array(lgm_sample_array)):
    Noise_Data_Test = np.array(df1['Oil Rate [bbl/day].1']) # Changed from 5 to 6 wei
    Model_Data_Test = np.array(lgm_sample_array[test_count])
    Std_Dev_Test = np.array(df3["MovingSD"])

    sum_count_test = np.sum((Noise_Data_Test - Model_Data_Test)**2   / (Std_Dev_Test)
    test_count = test_count +1
    ML_lgm_test_array.append(sum_count_test)
#print(ML_lgm_test_array)
```

Pan CRM

In [122…

```
len(np.array(df1['Oil Rate [bbl/day].1']))
```

Out[122… 85

In [123…

```
len(np.array(df3["MovingSD"]))
```

Out[123… 85

In [124…

```
len(np.array(panCRM_sample_array[0]))
```

Out[124… 85

In [125…

```
len(df3["MovingSD"])
```

Out[125… 85

In [126…

```
test_count = 0
ML_panCRM_test_array = []
while test_count< len(np.array(panCRM_sample_array)):
    Noise_Data_Test = np.array(df1['Oil Rate [bbl/day].1']) # Changed from 5 to 6 wei
    Model_Data_Test = np.array(panCRM_sample_array[test_count]) # Changed from 1 to 2
    Std_Dev_Test = np.array(df3["MovingSD"])

    sum_count_test = np.sum((Noise_Data_Test - Model_Data_Test)**2   / (Std_Dev_Test)
    test_count = test_count +1
    ML_panCRM_test_array.append(sum_count_test)
#print(ML_panCRM_test_array)
```

Probability Calculation

In [127…

```
len(ML_panCRM_test_array)
```

Out[127… 10000

In [128…

```
ML_df = pd.DataFrame({'SEM':ML_sem_test_array, 'Arpes':ML_arpes_test_array , 'LGM':ML
```

In [129…

```
ML_df
```

Out[129…

	SEM	Arpes	LGM	Pan CRM
0	1413.279272	1605.610390	1494.674371	1413.996073
1	1251.850358	1447.472687	1334.070026	1253.150079
2	1191.183525	1390.042389	1274.508556	1192.850241
3	1147.423776	1348.419511	1231.776616	1148.988913
4	1107.263600	1310.321696	1192.594952	1108.709026
...	...	...	...	...
9995	1290.391956	1596.012313	1386.228501	1288.302382
9996	1339.343897	1644.742954	1434.649574	1337.033747
9997	1399.572958	1704.650670	1494.132299	1397.093640
9998	1483.733877	1785.871880	1576.095614	1481.849005
9999	1670.379024	1960.294318	1754.268529	1668.258420

10000 rows × 4 columns

Test for 2d array (WORKING) [THIS NEEDS TO BE REMOVED LATER]

In [130…

```
df4=pd.DataFrame(columns=ML_df.columns)
for i in ML_df:
  df4[i]= ML_df[i]+ML_df['SEM']
  df4[i]+=ML_df[i]+ML_df['Arpes']
  df4[i]+= ML_df[i]+ML_df['LGM']
  df4[i]+= ML_df[i]+ML_df['Pan CRM']
```

In [131…

```
df4
```

Out[131…

	SEM	Arpes	LGM	Pan CRM
0	11580.677193	12350.001668	11906.257589	11583.544397
1	10293.944579	11076.433897	10622.823252	10299.143465
2	9813.318812	10608.754266	10146.618936	9819.985674
3	9466.303918	10270.286859	9803.715278	9472.564467
4	9147.943674	9960.176056	9489.269082	9153.725376
...	...	...	...	...
9995	10722.502976	11944.984405	11105.849155	10714.144680
9996	11113.145760	12334.741988	11494.368469	11103.905161
9997	11593.741399	12814.052249	11971.978765	11583.824130
9998	12262.485884	13471.037894	12631.932830	12254.946394
9999	13734.716388	14894.377565	14070.274409	13726.233970

10000 rows × 4 columns

Ends here

In [132…

```
# Look at this one more time. especially the last line (The code works but we need to
'''
df4=pd.DataFrame(columns=ML_df.columns)
for i in ML_df:
  df4[i]= (np.exp((-1/2)*(ML_df[i] - ML_df['SEM'])))
  df4[i]+=(np.exp((-1/2)*(ML_df[i] - ML_df['Arpes'])))
  df4[i]+= (np.exp((-1/2)*(ML_df[i] - ML_df['LGM'])))
  df4[i]+= (np.exp((-1/2)*(ML_df[i] - ML_df['Pan CRM'])))
  df4[i] = 1/(Number_of_Samples * df4[i])  # Confirm the denominator it was orignally

  '''
```

Out[132…

```
"\ndf4=pd.DataFrame(columns=ML_df.columns)\nfor i in ML_df:\n  df4[i]= (np.exp((-1/2)
*(ML_df[i] - ML_df['SEM'])))\n  df4[i]+=(np.exp((-1/2)*(ML_df[i] - ML_df['Arpes'])))
\n  df4[i]+= (np.exp((-1/2)*(ML_df[i] - ML_df['LGM'])))\n  df4[i]+= (np.exp((-1/2)*(M
L_df[i] - ML_df['Pan CRM'])))\n  df4[i] = 1/(Number_of_Samples * df4[i])  # Confirm t
he denominator it was orignally 10 but I changed it for a bigger sample size\n  \n  "
```

In [133…

```
porb_count_sem = np.zeros(len(df4))
df4=pd.DataFrame(columns=ML_df.columns)
for i in ML_df:
  df4[i]= (np.exp((-1/2)*(ML_df[i] - ML_df['SEM'])))
  #print(df4[i])
  porb_count_sem = df4[i] + porb_count_sem
#print(porb_count_sem)
porb_count_sem = porb_count_sem * len(df4)
porb_count_sem = 1 / porb_count_sem
porb_count_sem
```

Out[133…

```
0       0.000059
1       0.000066
2       0.000070
3       0.000069
4       0.000067
          ...
9995    0.000026
9996    0.000024
9997    0.000022
9998    0.000028
9999    0.000026
Length: 10000, dtype: float64
```

In [134…

```
porb_count_arpes = np.zeros(len(df4))
df4=pd.DataFrame(columns=ML_df.columns)
for i in ML_df:
  df4[i]= (np.exp((-1/2)*(ML_df[i] - ML_df['Arpes'])))
  #print(df4[i])
  porb_count_arpes = df4[i] + porb_count_arpes
#print(porb_count_arpes )
porb_count_arpes = porb_count_arpes * len(df4)
porb_count_arpes = 1 / porb_count_arpes
porb_count_arpes
```

Out[134…

```
0       1.013183e-46
1       2.181235e-47
2       4.587958e-48
3       1.551681e-48
4       5.428022e-49
            ...
9995    1.123921e-71
9996    1.155734e-71
9997    1.271815e-71
9998    6.908083e-71
```

```
9999    2.857933e-68
Length: 10000, dtype: float64
```

In [135…

```
porb_count_lgm = np.zeros(len(df4))
df4=pd.DataFrame(columns=ML_df.columns)
for i in ML_df:
  df4[i]= (np.exp((-1/2)*(ML_df[i] - ML_df['LGM'])))
  #print(df4[i])
  porb_count_lgm = df4[i] + porb_count_lgm
#print(porb_count_lgm)
porb_count_lgm = porb_count_lgm * len(df4)
porb_count_lgm = 1 / porb_count_lgm
porb_count_lgm
```

Out[135…

```
0       1.244911e-22
1       9.199780e-23
2       5.616585e-23
3       3.307390e-23
4       1.989203e-23
            ...
9995    4.024498e-26
9996    4.831206e-26
9997    6.575176e-26
9998    2.464308e-25
9999    1.563089e-23
Length: 10000, dtype: float64
```

In [136…

```
porb_count_panCRM = np.zeros(len(df4))
df4=pd.DataFrame(columns=ML_df.columns)
for i in ML_df:
  df4[i]= (np.exp((-1/2)*(ML_df[i] - ML_df['Pan CRM'])))
  #print(df4[i])
  porb_count_panCRM = df4[i] + porb_count_panCRM
#print(porb_count_panCRM)
porb_count_panCRM = porb_count_panCRM * len(df4)
porb_count_panCRM = 1 / porb_count_panCRM
porb_count_panCRM
```

Out[136…

```
0       0.000041
1       0.000034
2       0.000030
3       0.000031
4       0.000033
          ...
9995    0.000074
9996    0.000076
9997    0.000078
9998    0.000072
9999    0.000074
Length: 10000, dtype: float64
```

In [137…

```
P_df = pd.DataFrame({'SEM':porb_count_sem, 'Arpes':porb_count_arpes , 'LGM':porb_cour
```

In [138…

```
P_df
```

Out[138…

	SEM	Arpes	LGM	Pan CRM
0	0.000059	1.013183e-46	1.244911e-22	0.000041
1	0.000066	2.181235e-47	9.199780e-23	0.000034
2	0.000070	4.587958e-48	5.616585e-23	0.000030
3	0.000069	1.551681e-48	3.307390e-23	0.000031
4	0.000067	5.428022e-49	1.989203e-23	0.000033

...	...	...	...	...
9995	0.000026	1.123921e-71	4.024498e-26	0.000074
9996	0.000024	1.155734e-71	4.831206e-26	0.000076
9997	0.000022	1.271815e-71	6.575176e-26	0.000078
9998	0.000028	6.908083e-71	2.464308e-25	0.000072
9999	0.000026	2.857933e-68	1.563089e-23	0.000074

10000 rows × 4 columns

In [139…

```
P_df.sum(axis = 0, skipna = True)
```

Out[139…

```
SEM         6.082972e-01
Arpes       1.304238e-46
LGM         4.825889e-22
Pan CRM     3.917028e-01
dtype: float64
```

In [140…

```
Summed_P = P_df.sum(axis = 0, skipna = True)
```

In [141…

```
np.sum(Summed_P)
```

Out[141…

```
1.0
```

In [142…

```
np.sort(Summed_P)[::-1]
```

Out[142…

```
array([6.08297243e-01, 3.91702757e-01, 4.82588892e-22, 1.30423799e-46])
```

In [ ]:

```
```

Implementing the AIC Method

In [143…

```
len(qt_arpes_curve_fitted)
```

Out[143…

```
85
```

In [144…

```
len(df1['Oil Rate [bbl/day].1'])
```

Out[144…

```
85
```

In [145…

```
df_of_models = pd.DataFrame({'Model': ['sem', 'arpes', 'lgm', 'panCRM'],
                              'Raw_Noise': [df1['Oil Rate [bbl/day].1'], df1['Oil Rate
                         'Data': [qt_sem_curve_fitted, qt_arpes_curve_fitted, qt_lgm_curve_fit
```

In [146…

```
df_of_models
```

Out[146…

	Model	Raw_Noise	Data
0	sem	10 125.903226 11 120.466667 12 124.41...	10 104.052342 11 100.777758 12 97.62...
1	arpes	10 125.903226 11 120.466667 12 124.41...	10 140.723032 11 129.172995 12 119.22...
2	lgm	10 125.903226 11 120.466667 12 124.41...	10 121.517931 11 115.235070 12 109.42...

3	panCRM	10 125.903226 11 120.466667 12 124.41...	3 296.666883 4 264.684632 5 241.71...	

In [147…

```
arr1 = np.array(df_of_models['Raw_Noise'])
```

In [148…

```
arr2 = np.array(df_of_models['Data'])
```

In [149…

```
index = 0
summed = []
while index < 4:
    diff = (arr1[index] - arr2[index])**2
    index = index + 1
    summed.append(np.sum(diff))
```

In [150…

```
df_of_models["Squared Sum"] = summed
```

In [151…

```
df_of_models
```

Out[151…

	Model	Raw_Noise	Data	Squared Sum
0	sem	10 125.903226 11 120.466667 12 124.41...	10 104.052342 11 100.777758 12 97.62...	67073.308187
1	arpes	10 125.903226 11 120.466667 12 124.41...	10 140.723032 11 129.172995 12 119.22...	79378.642364
2	lgm	10 125.903226 11 120.466667 12 124.41...	10 121.517931 11 115.235070 12 109.42...	70680.336969
3	panCRM	10 125.903226 11 120.466667 12 124.41...	3 296.666883 4 264.684632 5 241.71...	10117.554275

In [152…

```
df_of_models["Squared Sum"][0]
```

Out[152…

```
67073.30818715402
```

AIC Calculation

In [153…

```
def aic(Data, Model, index, name):
    N = len(Data)
    ss = Model["Squared Sum"][index]
    k = len(signature(name).parameters) + 1

    AIC = N * (math.log(ss/N)) + 2*k
    return AIC
```

In [154…

```
aic(df1['Oil Rate [bbl/day].1'], df_of_models, 0, sem)
```

Out[154…

```
577.0256666414514
```

In [155…

```
aic(df1['Oil Rate [bbl/day].1'], df_of_models, 1, arpes)
```

Out[155…

```
591.3433361701248
```

In [156…

```
aic(df1['Oil Rate [bbl/day].1'], df_of_models, 2, lgm)
```

Out[156… 581.4780721838595

In [157…
```
aic(df1['Oil Rate [bbl/day].1'], df_of_models, 3, panCRM)
```

Out[157… 418.2469586970858

In [158…
```
index2 = 0
model_array = [sem, arpes, lgm, panCRM]
aic_array = []

while index2 < len(model_array):
    aic_calculated = aic(df1['Oil Rate [bbl/day].1'], df_of_models, index2, model_arr
    aic_array.append(aic_calculated)
    index2 = index2 +1
```

In [159…
```
df_of_models["AIC"] = aic_array
```

In [160…
```
df_of_models
```

Out[160…

	Model	Raw_Noise	Data	Squared Sum	AIC
0	sem	10 125.903226 11 120.466667 12 124.41...	10 104.052342 11 100.777758 12 97.62...	67073.308187	577.025667
1	arpes	10 125.903226 11 120.466667 12 124.41...	10 140.723032 11 129.172995 12 119.22...	79378.642364	591.343336
2	lgm	10 125.903226 11 120.466667 12 124.41...	10 121.517931 11 115.235070 12 109.42...	70680.336969	581.478072
3	panCRM	10 125.903226 11 120.466667 12 124.41...	3 296.666883 4 264.684632 5 241.71...	10117.554275	418.246959

Corrected AIC

In [161…
```
def caic(array, model):
    N = len(array)
    ss = SumSquares(array)
    k = len(signature(model).parameters) + 1

    AIC = N * (math.log(ss/N)) + 2*k

    CAIC = AIC + (2*k*(k+1))/(N-k-1)
    return CAIC
```

In [162…
```
def caic(Data, Model, index, name):
    N = len(Data)
    ss = Model["Squared Sum"][index]
    k = len(signature(name).parameters) + 1

    AIC = N * (math.log(ss/N)) + 2*k
    CAIC = AIC + (2*k*(k+1))/(N-k-1)

    return CAIC
```

In [163…
```
index3 = 0
model_array = [sem, arpes, lgm, panCRM]
caic_array = []
```

```
while index3 < len(model_array):
    caic_calculated = caic(df1['Oil Rate [bbl/day].1'], df_of_models, index3, model_a
    caic_array.append(caic_calculated)
    index3 = index3 +1
```

In [164…

```
df_of_models["CAIC"] = caic_array
```

In [165…

```
df_of_models
```

Out[165…

	Model	Raw_Noise	Data	Squared Sum	AIC	CAIC
0	sem	10 125.903226 11 120.466667 12 124.41...	10 104.052342 11 100.777758 12 97.62...	67073.308187	577.025667	577.785160
1	arpes	10 125.903226 11 120.466667 12 124.41...	10 140.723032 11 129.172995 12 119.22...	79378.642364	591.343336	592.102830
2	lgm	10 125.903226 11 120.466667 12 124.41...	10 121.517931 11 115.235070 12 109.42...	70680.336969	581.478072	582.237566
3	panCRM	10 125.903226 11 120.466667 12 124.41...	3 296.666883 4 264.684632 5 241.71...	10117.554275	418.246959	419.323882

Sorting with AIC

In [166…

```
df_of_models.sort_values(by=['AIC'])
```

Out[166…

	Model	Raw_Noise	Data	Squared Sum	AIC	CAIC
3	panCRM	10 125.903226 11 120.466667 12 124.41...	3 296.666883 4 264.684632 5 241.71...	10117.554275	418.246959	419.323882
0	sem	10 125.903226 11 120.466667 12 124.41...	10 104.052342 11 100.777758 12 97.62...	67073.308187	577.025667	577.785160
2	lgm	10 125.903226 11 120.466667 12 124.41...	10 121.517931 11 115.235070 12 109.42...	70680.336969	581.478072	582.237566
1	arpes	10 125.903226 11 120.466667 12 124.41...	10 140.723032 11 129.172995 12 119.22...	79378.642364	591.343336	592.102830

Sorting with CAIC

In [167…

```
df_of_models.sort_values(by=['CAIC'])
```

Out[167…

	Model	Raw_Noise	Data	Squared Sum	AIC	CAIC
3	panCRM	10 125.903226 11 120.466667 12 124.41...	3 296.666883 4 264.684632 5 241.71...	10117.554275	418.246959	419.323882
0	sem	10 125.903226 11 120.466667 12 124.41...	10 104.052342 11 100.777758 12 97.62...	67073.308187	577.025667	577.785160
2	lgm	10 125.903226 11 120.466667 12 124.41...	10 121.517931 11 115.235070 12 109.42...	70680.336969	581.478072	582.237566
1	arpes	10 125.903226 11 120.466667 12 124.41...	10 140.723032 11 129.172995 12 119.22...	79378.642364	591.343336	592.102830