AMERICAN UNIVERSITY OF BEIRUT

WELL TRAJECTORY DESIGN WITHIN THE INTEGRATED FIELD DEVELOPMENT OPTIMIZATION FRAMEWORK

by
ANTHONY JEAN AYOUB

A thesis
submitted in partial fulfillment of the requirements
for the degree of Master of Science
to the Department of Chemical Engineering
of the Maroun Semaan Faculty of Engineering and Architecture
at the American University of Beirut

Beirut, Lebanon
February 2021

AMERICAN UNIVERSITY OF BEIRUT

WELL TRAJECTORY DESIGN WITHIN THE INTEGRATED FIELD DEVELOPMENT OPTIMIZATION FRAMEWORK

by
ANTHONY JEAN AYOUB

Approved by:

Kassem Ghorayeb [illegible]

Advisor

Kassem Ghorayeb, on behalf of Dr. Elsa Maalouf [illegible]

Member of Committee

Kassem Ghorayeb, on behalf of Dr. Georges Saad [illegible]

Member of Committee

Kassem Ghorayeb, on behalf of Dr. Joseph Zaiter [illegible]

Member of Committee

Date of thesis defense: February 22, 2021

AMERICAN UNIVERSITY OF BEIRUT

THESIS RELEASE FORM

Student Name:	Ayoub	Anthony	Jean
	Last	First	Middle

I authorize the American University of Beirut, to: (a) reproduce hard or electronic copies of my thesis; (b) include such copies in the archives and digital repositories of the University; and (c) make freely available such copies to third parties for research or educational purposes:

☒ As of the date of submission

☐ One year from the date of submission of my thesis.

☐ Two years from the date of submission of my thesis.

☐ Three years from the date of submission of my thesis.

[illegible] 28/2/2021

Signature Date

ACKNOWLEDGEMENTS

Firstly, I would like to thank God for providing me with the strength, determination, and energy to be able to successfully finalize this thesis.

Secondly, I would like to thank my advisor Prof. Ghorayeb for his continuous support, encouragement, and guidance throughout this thesis. Due to this research project, I have gained a large amount of knowledge in the oil and gas field and specifically in drilling.

Thirdly, I would like to thank all members of the OGED team for providing assistance and support when needed. I am proud to call myself a member of this amazing and hardworking team.

Lastly, I would like to thank my family and friends for their support throughout this experience. I could not have done it without you!

ABSTRACT
OF THE THESIS OF

Anthony Jean Ayoub for Master of Science
Major: Chemical Engineering

Title: Well Trajectory Design within the Integrated Field Development Optimization Framework.

The drilling of vertical wells was the most used technique in the oil and gas industry. However, as time progressed, the need to drill in a non-vertical path increased in significance. This led to the creation of directional and horizontal drilling. Due to the improvement in technology, these two different methods of drilling have become widely used in the industry today (Short, 1993). The usage of a single directional or horizontal well may be as efficient as using multiple vertical wells. Therefore, the main advantage of such methods in drilling is that it can minimize drilling costs while maximizing profit.

In this thesis paper, we develop a Python tool aiming at 1) designing the well trajectory and 2) assessing the feasibility of the proposed reservoir targets as part of the integrated well and facility placement optimization solution being developed by the OGED team. The input to the "building block" we are developing is the horizontal section of the well as well as the surface location. The output of the algorithm is a well trajectory honoring the well design constraints.

We start by studying the different well design techniques, comparing their accuracy, and implementing these techniques in the tool. This discussion includes the different parameters related to the used well profiles. We apply the different methods to several well placement scenarios and test the resulting well trajectories within the framework of the integrated well and facility placement optimization solution to assess the impact on the efficiency and robustness of the solution as well as the associated NPV.

TABLE OF CONTENTS

ILLUSTRATIONS

TABLES

CHAPTER 1

INTRODUCTION

In this chapter, the background of well drilling will be discussed. Followed by the presentation of the problem statement of this paper. Lastly, the different objectives needed to finalize this study will be explained.

1.1. Background

Drilling of a well is simply the construction of a path or trajectory that connects two different points. The first point is the starting point of the trajectory. This point is commonly denoted by the drilling center. On the other hand, the second point is known as the reservoir target. This point can represent a reservoir or another well that needs to be intersected. In the oil and gas industry, the three different well types currently implemented are the vertical, directional, and horizontal wells. These types are characterized based on how the trajectory intersects the target. For the case of vertical wells, the planned trajectory intersects the target vertically. In the case of directional wells, the planned path intersects the target at a calculated angle. Lastly, in the case of horizontal wells, the planned path intersects the target horizontally.

For directional and horizontal wells, the most important parameter to be considered is the maximum allowable degree of bending or dogleg severity. Drilling engineers must take into consideration that the higher the dogleg severity, the larger the deviation, the smaller the well length required to intersect the target (Hosseini et al., 2014). However, having high levels of dogleg severity may lead to more frequent drilling complications (Short, 1993). Therefore, having the highest possible dogleg severity is not always the best option. A second important parameter to be discussed is the specification of the kick-off point. This point is simply the point at which the vertical section of the well starts deviating to intersect the target (Inglis, 1987). The depth of the kick-off point is normally based on the existing scenario. For example, a shallow kick off point is selected when large inclinations are needed to intersect the target. In other cases, a deep kick off point is selected in order to avoid "soft formations" that are near the surface (Inglis, 1987). Similarly, the usage of the different well profiles present in directional and horizontal wells are also dictated by the scenario present in each case (Devereux, 1998). After choosing the well profile that serves as the best fit, the drilling engineer then combines several equations to determine the trajectory. This is known as the designing phase.

Following the designing phase, the drilling engineer must ensure that the well is moving according to the correct path. In the designing phase, the 2D trajectory of the well path is obtained. Therefore, to obtain this trajectory in 3D space, surveying methods are implemented (Bourgoyne et al., 1991). These different methods are used to calculate the three-dimensional coordinates of the points that are found in the calculated well trajectory.

1.2. Problem Statement

The usage of directional and horizontal wells has proven to be beneficial when compared to vertical wells. However, the equations that are used in these well profiles do not account for any change in direction. To solve this issue, engineers usually refer to complex wells (which consist of different turn and build sections). Complex wells may be combined with different surveying techniques to generate the best well trajectory. Thus, the aim of this study is to design a complex well algorithm that can be optimized and implemented in the novel integrated solution being developed by the OGED team. This team, which is led by Professor Ghorayeb, is currently working on field development planning. Thus, it has been developing both well and facility placements to be integrated into the studied field development.

1.3. Objectives

1. Design and test a directional well algorithm after studying the different well profiles present in the literature.
2. Improve the directional well algorithm to account for different bends and/or turns.
3. Test the newly updated algorithm and visualize the results.
4. Modify and optimize the algorithm.
5. Implement the algorithm into the integrated solution being developed by the OGED team.
6. Test and verify the newly implemented algorithm using different scenarios.

CHAPTER 2

DEFINITIONS

This chapter will discuss the different important definitions that are implemented in well designing and drilling.

2.1. True Vertical Depth (TVD)

True vertical depth refers to the vertical depth with respect to a reference such as the surface. If a point lies 300 meters vertically below a surface, then this point's TVD is equal to 300 meters.

2.2. Measured Depth (MD)

Measured depth refers to the actual length from the reference to the point. Therefore, the curves and bends of the well are accounted for when using measured depth.

2.3. Horizontal Displacement

Horizontal displacement is the horizontal distance between the starting point of the well and the point of interest.

2.4. End of Build Point

The end of build point is the point at which the well stops deviating from vertical to reach the target. This point is normally obtained by the calculations used for the different well profiles present in the literature.

2.5. Vertical Section

This section simply represents the vertical part of the well. For example, for a vertical well the vertical section is equal to the length of the whole well being drilled. In the case of directional and horizontal wells, this section is normally limited by the kick off point.

2.6. Build Section

This section represents the part at which the well starts deviating from the vertical. Therefore, what is normally seen in this section is the change in the inclination and/or azimuth. The build section is limited by the kick off point and the end of build point.

2.7. Tangent Section

After having a build section, the well's inclination angle is usually kept constant. Therefore, this part of the well is known as the tangent section. It is important to note that, in this section, the TVD and horizontal displacement of the well is changing. This section is limited by the end of build point and/or target. Some of the following definitions are shown in the figure below:

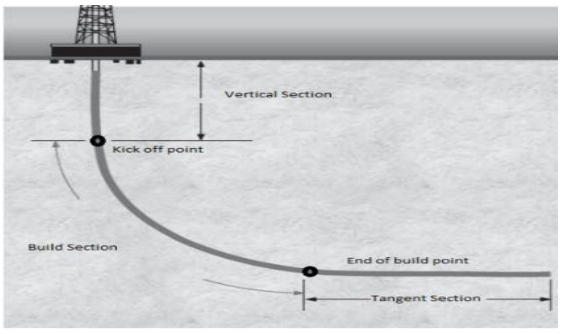


Figure 1 Different sections in a horizontal well

CHAPTER 3

LITERATRUE REVIEW

The usage of each well type is dictated by the scenario present in each case. Vertical wells were primarily used in the case when the target was at a specific vertical distance below the starting point. In this specific scenario, only vertical wells were needed to intersect the target. As the cases became more complex, the usage of directional and horizontal wells became of grave importance. The different applications, formulas, and examples for each well profile will be presented in this chapter.

3.1. Application for Directional and Horizontal wells

3.1.1. Main Applications of Directional and Horizontal Wells

- In many drilling scenarios, drilling engineers need to avoid certain geological formations for the procedure to be successful. This can be seen in the case of the presence of a salt dome above the targeted reservoir. In this specific case, using the traditional vertical well to intersect the target will yield to complications such as "corrosion" (Inglis, 1987). To add to the example above, another geological formation that must be avoided is the presence of "steep inclined faults" (Inglis, 1987).
- In offshore drilling, multiple wells are usually drilled from the same platform. In this scenario, wells being drilled must be able to intersect targets that are kilometres away. For this reason, using directional wells is almost mandatory

since it requires the usage of a single platform rather than multiple platforms when compared to vertical wells (McMillian, 1980).

- In other drilling cases, multiple targets are present that need to be intersected. Therefore, in order to minimize costs, drilling engineers will resort to a single directional well that will intersect the different targets rather than drilling multiple vertical wells for the same purpose (Hossain & Al-Majed, 2015).
- In other scenarios, directional wells are used to relief already present wells that are undergoing excessive pressures and flowrates. The aim of these directional wells (also known as relief wells) is to intersect the targeted wells and prevent them from blowing (Short, 1993).

3.1.2. Scenarios that Favor Horizontal Wells

- It has been proven that horizontal wells have higher production rates when compared to other well types (Short, 1993). This improvement in productivity is directly related to the long horizontal section of these types of wells. Therefore, as the horizontal section of the well increases, the area that is in common between the well and the reservoir increases. This in turn allows for higher volume of oil and gas to flow from the reservoir to the well (Short, 1993). As a result, certain scenarios that were proven to be economically unfeasible using the vertical or directional well type may be feasible using the horizontal well type.
- Horizontal wells are also used in the case when “vertical fractures” that contain hydrocarbons must be penetrated (Short, 1993).

- In addition, horizontal wells are also applied to avoid water and gas coning problems that arise when the pressure difference between the reservoir and the well increases (Hossain & Al-Majed, 2015)**.** This is possible due to the inversely proportional relation that exists between the contact area and the pressure difference. Therefore, as the contact area between the reservoir and the well increases, the pressure difference between the reservoir and the well will decrease (Short, 1993).

Horizontal wells are usually more expensive when compared to the other well types. Therefore, the usage of these wells must be economically justified.

3.2. Inclination and Build Rate

In addition to the terms defined in chapter 2, there exist other basic components that are used in well trajectory planning. These components will be discussed in this section.

Inclination is the angle that is being formed between the well and the vertical plane. As stated previously, a vertical well will intersect the target at an inclination of zero degrees. Horizontal wells intersect the target at an inclination of 90 degrees. The higher the well inclination, the greater the complications faced such as logging (Allain, et al., The Imperial College Lectures in Petroleum Engineering Drilling and Reservoir Appraisal, 2019). Moreover, the higher the inclination, the larger the costs associated with drilling (Ma et al., 2016). Therefore, the usage of inclinations above 60 degrees must be economically justified. Build rate is simply the change in inclination over a distance that could be 30 meters or 100 feet. The figure below is a graphical representation of inclination.

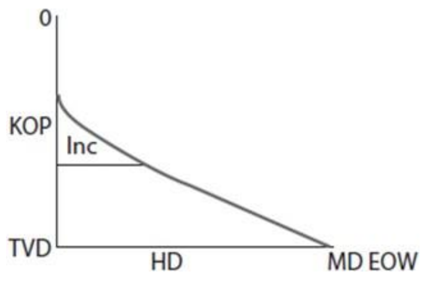


Figure 2 Well inclination

3.3. Azimuth and Turn Rate

Azimuth or direction is the angle formed with respect to the north when projected on the horizontal plane. It usually ranges between 0 and 360 degrees. By convention, it is always measured clockwise with respect to the north (Bourgoyne et al., 1991). Thus, an azimuth of 90 degrees indicates a turn towards the east. Moreover, an azimuth of 270 degrees indicates a turn towards the west. If the difference between two azimuths is positive, then the well must direct itself to the right. In the same manner, if the difference between the two angles is negative, then the well will turn to the left (McMillian, 1980)**.** Similarly to the build rate, the turn rate is simply the change in azimuth over a distance that could be 30 meters or 100 feet. The figure below illustrates the graphical representation of the azimuth.

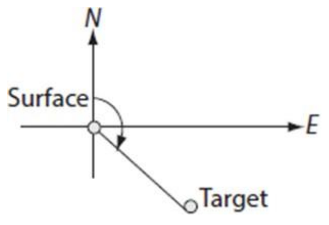


Figure 3 Well azimuth

3.4. Dogleg Severity and Dogleg Angle

As stated in the introduction, in directional and horizontal wells, drilling engineers are supposed to take into consideration the risk of having certain equipment such as the drill string (which is composed of the different drilling tools such as drill pipe etc...) stuck while drilling. This is also known as key seat (Short, 1993). In other cases, highly deviated wells may witness casing wear and fatigue (Devereux, 1998**.** Therefore, engineers usually set a constraint for the maximum allowable dogleg severity during drilling. Normally, the dogleg severity is calculated between two points known as "survey stations" (Hossain & Al-Majed, 2015). It has the following formula (Hossain & Al-Majed, 2015):

$$DLS = \frac{\Phi * 30}{MD}$$

Where Φ, and MD are the dogleg angle in degrees, and measured depth between the two points being studied respectively in meters. The unit for the dogleg severity (DLS) is degrees per 100 feet or 30 meters.

The maximum dogleg severity allowed usually depends on the depth of the well. In the case of shallow wells, a maximum dogleg severity of 10 degrees per 100 feet or 30 meters is allowable. On the other hand, in the case of deep wells, any dogleg severity exceeding 3 degrees per 100 feet or 30 meters may lead to drilling problems (Mitchell, 1995).

The formula used to obtain the dogleg angle is the following (Hossain & Al-Majed, 2015):

$$\Phi = cos^{-1}(Cos\ I1 * Cos\ I2 + Sin\ I1 * Sin\ I2 * Cos\ (A2 - A1)$$

Where, I1 and I2 represent the inclination angles at points 1 and 2 respectively. A1 and A2 represent the azimuth for each survey point (Hossain & Al-Majed, 2015).

For the purpose of this thesis paper, the only two well profiles that will be described in detail are the build and hold and the complex well profiles. The other different well profiles will be mentioned briefly.

3.5. Directional Drilling Well Profiles

Other well types do exist in the directional category. This may differ slightly from one source to the other, but the directional category contains between four or five well types. Thus, these well profiles are shown below:

- The first well profile is the build and hold (Inglis, 1987).
- The second well profile is the build, hold, and drop (Bourgoyne et al., 1991).
- The third well profile is the continuous build or J type profile (Krishnan & Kulkarni, 2016).
- The fourth well profile is the slant well profile (Short, 1993).
- Finally, the fifth well profile present in the literature is the extended reach well (Short, 1993).

3.5.1. Build and Hold or Single Bend

The build and hold trajectory is divided into three different well sections. The first section is the vertical section. In this section, the well is drilled vertically to reach the kick-off point. Once the kick-off point is reached, the well then starts deviating at a given build rate. This forms the build section of the well. Similarly to the vertical section, the well stops deviating once the end of build point is reached. Following this section, the tangent section then drives the well to intersect the target at a fixed inclination (known as the hold section).

The figure below shows a geometrical representation of the build and hold profile. The components of this figure will be discussed throughout this section.

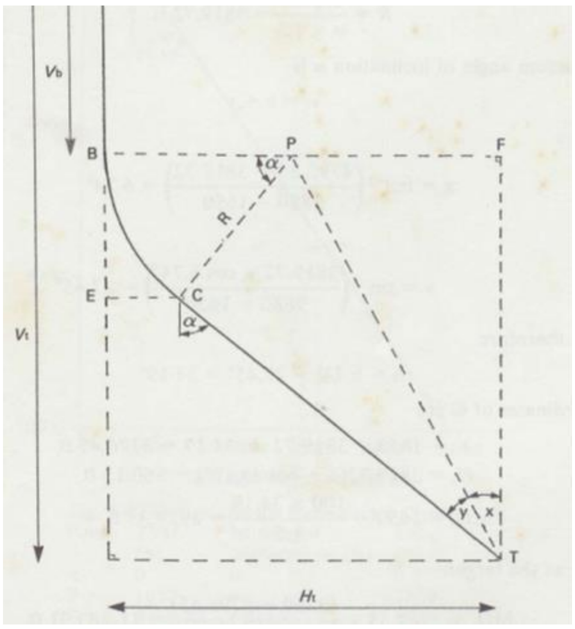


Figure 4 Build and hold profile.

To obtain the radius of curvature, which is represented by “R” in figure 5, the following equation is used (Bourgoyne et al., 1991):

$$R = \frac{180}{\pi * BR} = \frac{180 * MD}{\pi * BRA}$$

Where BR, BRA, and MD are the build rate, build rate angle, and measured depth respectively. Therefore, the build rate is simply the build rate angle per measured depth that may be either in feet or in meters (Bourgoyne et al., 1991). After the build rate is calculated, the horizontal displacement between the target and starting point must be determined. In figure 5, the horizontal displacement is represented by “Ht”. To calculate “Ht”, the following equation is used (Inglis, 1987):

$$Ht = \sqrt{(Xt - Xsp)^2 + (Yt - Ysp)^2}$$

Where Xt, Yt, Xsp, and Ysp represent the x and y coordinates of the target and starting point respectively (Inglis, 1987).

Following the horizontal displacement, the engineer needs to obtain the inclination required to reach the target. The equation used for that purpose is found below (Hossain & Al-Majed, 2015):

$$\alpha = \sin^{-1}\left[\frac{R}{\sqrt{(R-Ht)^2 + (TVD-KOP)^2}}\right] - \tan^{-1}\left[\frac{R-Ht}{TVD-KOP}\right]$$

Where α and TVD are the inclinations and true vertical depth of the target respectively.

These calculations will provide the needed components to guide the well towards the target. However, the direction of the well with respect to the horizontal plane or azimuth should be calculated. The following equation is used in the literature to calculate the direction (Inglis, 1987):

$$\beta = \tan^{-1}\left(\frac{Yt-Ysp}{Xt-Xsp}\right)$$

This equation can be obtained simply by specifying the angle that is being formed with the horizontal plane. Once this angle is specified, the application of the tangent rule is used to calculate this angle based on the given information.

Once all components have been calculated, the directional engineer must calculate the measured depth to intersect the target. Therefore, the following equations are used (Bourgoyne Jr., Millheim, Chenevert, & Jr., Applied Drilling Engineering, 1991):

$$MD_{T} = MD_{EOB} + MD_{TS}$$

$$MD_{EOB} = MD_{KOP} + 30 * \frac{\alpha}{BR}$$

Where, MDT, MDEOB, MDTS, MDKOP, α and BR are the measured depth of the target, end of build point, tangent section of the well, kick off point, inclination and build rate respectively.

3.5.2. Build, Hold, and Drop or S profile.

This well profile is similar to the build and hold profile discussed in 3.5.1. Thus, it includes the typical vertical section, build section and hold section. However, what differentiates this well type is the addition of a fourth section. This is known as the drop section. Therefore, the well will intersect the target at a decreased inclination that in some cases may even reach vertical or 0 degrees. The geometrical representation of this well type is presented in the figure below:

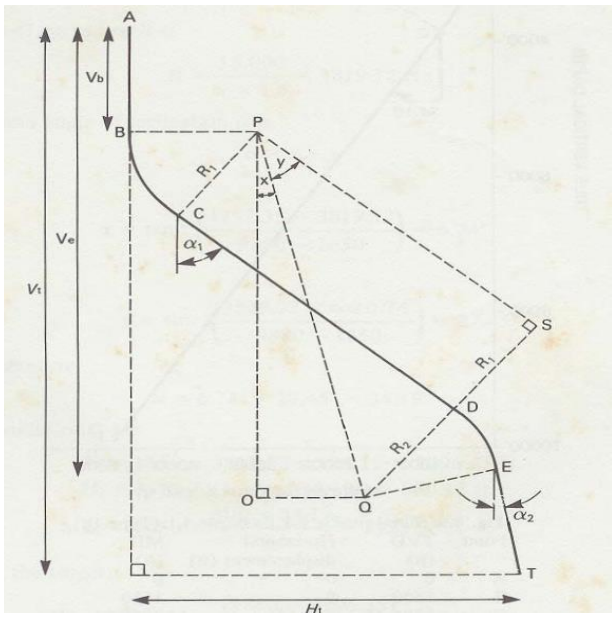


Figure 5 Build, hold, and drop profile.

In this configuration, there are two radii of curvature which are designated by “R1” and “R2”. “R1” is related to the build section and “R2” is related to the drop section. Thus, there exists two different equations based on the build rate and drop rate of each well section. These two radii of curvature are calculated by the following equations (Bourgoyne Jr., Millheim, Chenevert, & Jr., Applied Drilling Engineering, 1991):

$$\text{R1} = \frac{180}{\pi * BR} = \frac{180 * MD}{\pi * BRA}$$

$$\text{R2} = \frac{180}{\pi * DR} = \frac{180 * MD}{\pi * DRA}$$

Where BR, BRA, DR, DRA and MD are the build rate, build rate angle, drop rate, drop rate angle, and measured depth respectively.

To obtain the inclination required to reach the target, the directional driller must use the following formulas (Bourgoyne et al., 1991):

If R1+R2>Ht:

$$\alpha 1 = \tan^{-1}\left[\frac{TVD-KOP}{(R1+R2-Ht)}\right] - \cos^{-1}\left(\left[\frac{R1+R2}{TVD-KOP}\right] * sin\left[\tan^{-1}\left[\frac{TVD-KOP}{(R1+R2-Ht)}\right]\right]\right)$$

If R1+R2<Ht:

$$\alpha 1 = 180 - \tan^{-1}\left[\frac{TVD-KOP}{(Ht-(R1-R2)}\right] - \cos^{-1}\left(\left[\frac{R1+R2}{TVD-KOP}\right] * sin\left[\tan^{-1}\left[\frac{TVD-KOP}{(Ht-(R1+R2)}\right]\right]\right)$$

Note: In both cases presented above, the final inclination needed to intersect the target is at 0 degrees.

3.5.3. Continuous Build or J Profile

This well profile consists of two main sections. The first section is the vertical section where the well displaces vertically to reach the kick-off point. Once it reaches the kick-off point, the well then deviates to intersect the target which forms the build section. Therefore, there is no hold or drop sections in this well profile. It is important to state that, in this section, the kick-off point is normally located at very deep depths when compared to the previous well profiles. A graphical representation of this well profile is found in the figure below:

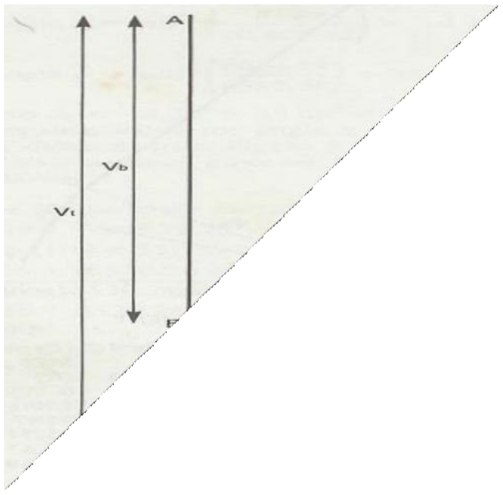


Figure 6 Continuous build

All parameters present in this well profile are calculated using the same formulas presented in section 3.5.1.

3.5.4. Slant Profile

In this well profile, the well starts with an inclination that ranges between 30 to 45 degrees (Short, 1993). Thus, there is no vertical section to reach the kick-off point as seen in figure 7. The well then continues with the same inclination to intersect the target. This is known as the hold section or tangent section. Moreover, all parameters in this well section are obtained by the equations presented in the sections above.

3.5.5. Extended Reach Profile

It is important to note that, in some other sources, the extended reach well profile is considered as a horizontal well type. For example, according to Hossain and

Al-Majed (2015), the extended reach well type is a special case of horizontal drilling. This is also the case according to Inglis (1987). For the purpose of this paper, the different well profiles will follow the classifications presented in section 3.5. This can be explained by the fact that the extended reach well has an almost identical shape as the build and hold profile. Therefore, it is more logical to include this well profile in the directional well's category. This can be seen in the figure below:

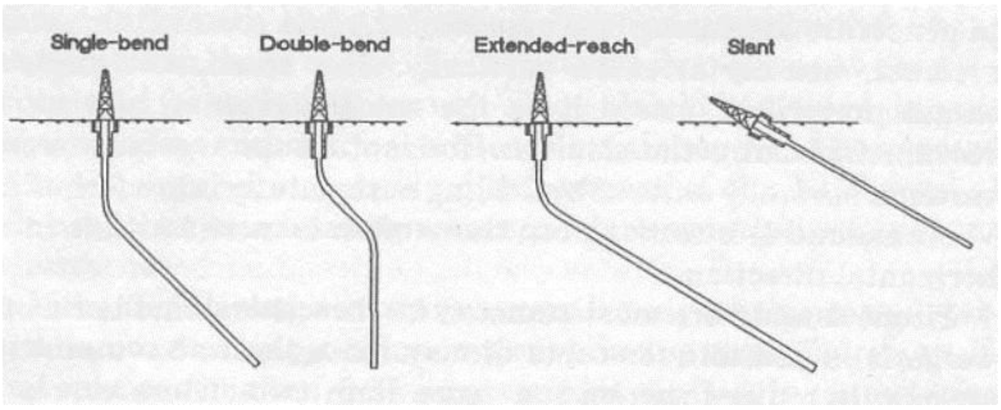


Figure 7 Different well profiles present in directional drilling.

Hence, it is evident that the extended reach profile is similar to the build and hold but with a greater inclination angle to reach the target that may sometimes reach 90 degrees.

3.6. Horizontal Drilling

Horizontal drilling is considered a special category of directional drilling (Hossain & Al-Majed, 2015). In this category, wells will intersect the target at an inclination of 90 degrees. Horizontal wells are divided into three main well profiles. The radius of curvature is what characterizes the different well profiles in this well category. It is divided into the following:

- Long radius or turn
- Medium radius or turn
- Short radius or turn

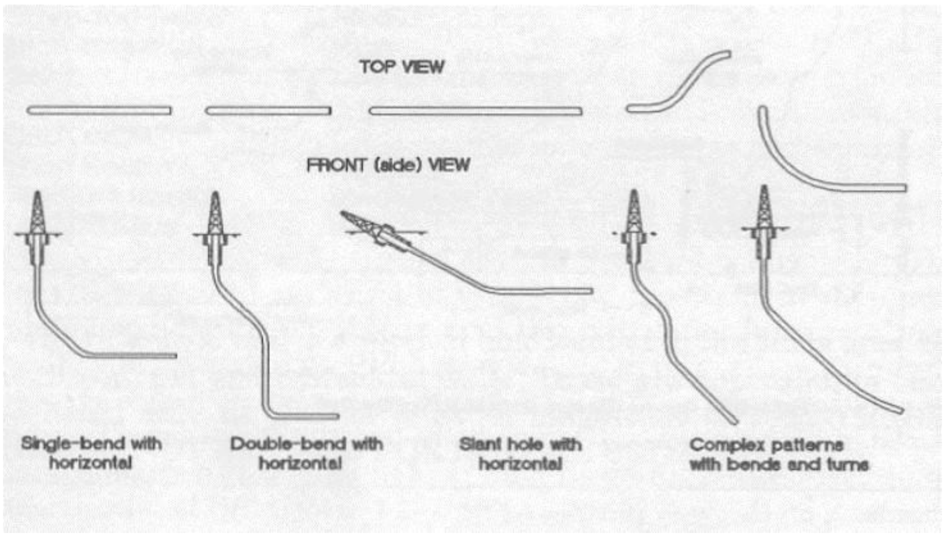
All three categories mentioned above have the same well sections. The first well section is the vertical section as described in section 3.5.1. The second well section is the build section. In this section, the well will build until it reaches an inclination of 90 degrees. Once this inclination is reached, the build section is terminated, and the well continues on horizontally to intersect the target forming the tangent or hold section. The main difference between the three categories is presented in the table below:

Table 1 Different horizontal well profiles.

	Long Radius	**Medium Radius**	**Short Radius**
Build Rate	2-6 degrees per 100 ft or 30 meters.	8-30 degrees per 100 ft or 30 meters.	60-150 degrees per 100 ft or 30 meters.
Radius of Curvature	304-914 meters.	91-243 meters.	0.6-18 meters.
Horizontal Section (after reaching 90 degree inclination).	609-1524 meters.	457-914 meters.	30-243 meters.

3.7. Complex Wells

Complex wells were first considered as futuristic approaches to solve modern day problems. The implementation of such well profiles was directly related to the level of advancement in technologies related to drilling. These types of wells do add a certain degree of complexity in the drilling process. However, with the development of enhanced drilling equipment, the usage of these types of wells has become relatively common. Complex wells are wells that usually contain many changes in direction. Moreover, these wells may consist of different build and drop sections (Short, 1993). In other cases, complex wells may even combine between directional and horizontal wells (Short, 1993). Examples of different complex wells are illustrated in the figure below:


TOP VIEW
FRONT (side) VIEW
Single-bend with horizontal
Double-bend with horizontal
Slant hole with horizontal
Complex patterns with bends and turns
Figure 8 Different complex wells.

Other examples of complex wells are present in the literature. According to Allain et al. (2019), the hook well and the snake well are examples of wells present in the industry today. Examples of the two wells will be presented in the two figures below:

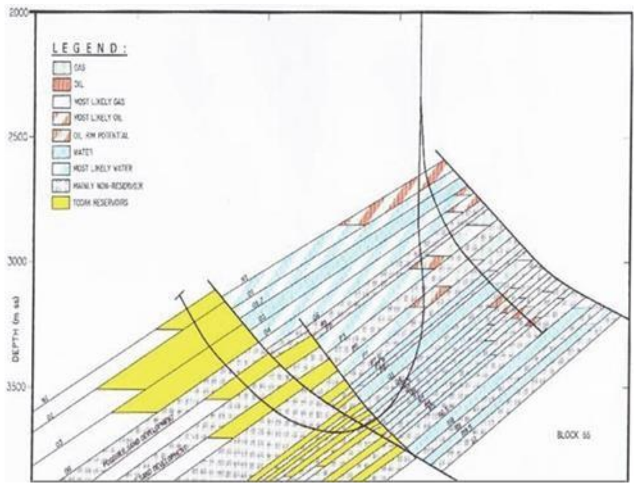


Figure 9 Hook well profile.

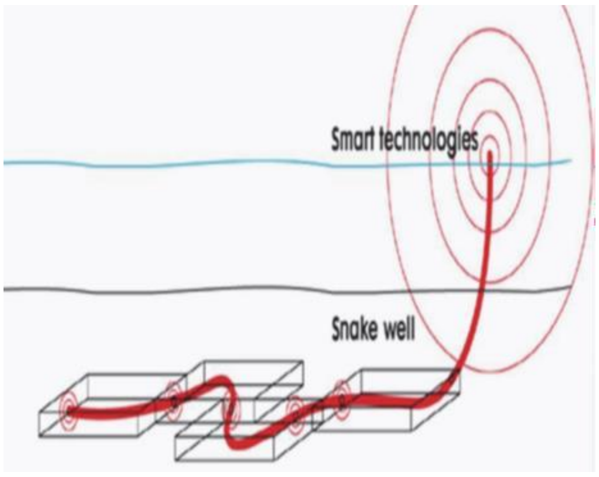


Figure 10 Snake well profile.

As seen in figure 9, the hook well may be used in the specific case when the well trajectory must go below a certain obstacle such as a fault and then rise to intersect the target. On the other hand, in figure 10, the snake well profile may be used in the case of multiple targets. Therefore, instead of using multiple vertical wells or even directional wells to intersect the targets, the drilling engineer may be able to use one single complex well. As a result, any well profile with bends and turns can be classified as a complex well.

As stated previously, the main disadvantage related to directional and horizontal wells is the assumption that no change in direction is required to intersect the target. Therefore, the build and hold profile assumes the need of a straight line to intersect the target. However, in real cases witnessed in the oil and gas industry, directional engineers most likely will be required to drive the well in a path full of bends and turns. To be able to design this 3D curved trajectory, engineers resort to different surveying techniques which require the usage of survey points. According to Bourgoyne et al. (1991), for each point, the inclination, azimuth, and measured depth between the two points are required to successfully perform the needed calculations.

3.8. Surveying Techniques

The different surveying techniques are mainly divided into two different parts. The first part assumes that a straight line can join the two survey points being studied (Bourgoyne et al., 1991). While as, for the second part, the main assumption is that a curve can join the two survey points used (Bourgoyne et al., 1991). The five different methods used are the following (Inglis, 1987):

1. Tangential method.
2. Balanced tangential method.
3. Average angle method.
4. Radius of curvature method.
5. Minimum curvature method.

3.8.1. Tangential Method

In this surveying method, the path of the well is assumed to follow a straight line. It is solely defined by the inclination and azimuth of the lower survey point (Inglis, 1987). The equations used in this method are the following (Inglis, 1987):

$$\Delta \mathrm{V} = \mathrm{L} \,.\, \cos \mathrm{I2}$$

$$\Delta \mathrm{N} = \mathrm{L} \,.\, \sin \mathrm{I2} \,.\, \cos \mathrm{A2}$$

$$\Delta \mathrm{E} = \mathrm{L} \,.\, \sin \mathrm{I2} \,.\, \sin \mathrm{A2}$$

Where I, A, and L are inclination, azimuth, and measured depth between the survey points respectively. The numbers 1 and 2 represent the previous and next station points to be calculated.

However, this calculation method is not recommended since it may lead to large errors when the well path changes.

3.8.2. Balanced Tangential Method

This surveying methods assumes that the actual path between the two survey points can be measured by the usage of two equal straight lines (Hossain & Al-Majed, 2015). Based on this surveying technique, the well path is defined by the inclination and azimuth of both surveying points (Hossain & Al-Majed, 2015). The equations used in this method are the following (Hossain & Al-Majed, 2015):

$$\Delta \mathrm{V} = \frac{L}{2} \,.\, (\cos \mathrm{I1} + \cos \mathrm{I2})$$

$$\Delta \mathrm{N} = \frac{\mathrm{L}}{2} \,.\, (\sin \mathrm{I1} \,.\, \cos \mathrm{A1} + \sin \mathrm{I2} \,.\, \cos \mathrm{A2})$$

$$\Delta \mathrm{E} = \frac{L}{2} \,.\, (\sin \mathrm{I1} \,.\, \sin \mathrm{A1} + \sin \mathrm{I2} \,.\, \sin \mathrm{A2})$$

Where I, A, and L are inclination, azimuth, and measured depth between the survey points respectively. The numbers 1 and 2 represent the previous and next station points to be calculated.

3.8.3. Average Angle Method

In this surveying method, the path of the well is assumed to follow a straight line that joins the two survey points (Hossain & Al-Majed, 2015). The inclination and azimuth of both points are averaged (Hossain & Al-Majed, 2015). The equations used in this method are the following:

$$\Delta V = L \,.\cos\left(\frac{I1 + I2}{2}\right)$$

$$\Delta N = L \,.\sin\left(\frac{I1 + I2}{2}\right).\cos\left(\frac{A1 + A2}{2}\right)$$

$$\Delta E = L \,.\sin\left(\frac{I1 + I2}{2}\right).\sin\left(\frac{A1 + A2}{2}\right)$$

Where I, A, and L are inclination, azimuth, and measured depth between the survey points respectively. The numbers 1 and 2 represent the previous and next station points to be calculated.

3.8.4. Radius of Curvature Method

In this surveying method, the path that connects the two surveying points is assumed to follow a circular arc (Hossain & Al-Majed, 2015). This circular arc is tangential to the inclination and azimuth at the surveying points (Inglis, 1987). The equations used in this surveying method are as follows (Hossain & Al-Majed, 2015):

$$\mathrm{Rv} = \frac{\mathrm{L}}{\mathrm{I2} - \mathrm{I1}} \cdot \frac{180}{\pi}$$

$$\Delta\mathrm{V} = \mathrm{Rv} \,.\, (\sin \mathrm{I2} - \sin \mathrm{I1})$$

$$\Delta\mathrm{H} = \mathrm{Rv}\,.\,(\cos \mathrm{I1} - \cos \mathrm{I2})$$

$$\Delta\mathrm{N} = \frac{\mathrm{L}}{\alpha 2 - \alpha 1} \cdot \left(\frac{180}{\pi}\right)^2 \cdot \frac{(\cos \mathrm{I1} - \cos \mathrm{I2})(\sin A2 - \sin A1)}{\mathrm{A2} - \mathrm{A1}}$$

$$\Delta\mathrm{N} = \frac{\mathrm{L}}{\alpha 2 - \alpha 1} \cdot \left(\frac{180}{\pi}\right)^2 \cdot \frac{(\cos \mathrm{I1} - \cos \mathrm{I2})(\cos A1 - \cos A2)}{\mathrm{A2} - \mathrm{A1}}$$

Where Rv, L, I, and A are the radius in the vertical plane, measured depth between the survey points, inclination, and azimuth respectively. The numbers 1 and 2 represent the previous and next station points to be calculated.

3.8.5. Minimum Curvature Method

This method is considered as an extension to the balanced tangential method (Inglis, 1987). In the case of the minimum curvature methood, the path between two points is estimated by the usage of a curve (Inglis, 1987). This curvature is obtained by the usage of a ratio factor or “F” (Inglis, 1987). The equations used in this method are presented below (Inglis, 1987):

$$\mathrm{F} = \frac{2}{\Phi}\left(\frac{180}{\pi}\right) \tan\left(\frac{\Phi}{2}\right)$$

$$\Delta\mathrm{V} = \mathrm{F}\,\frac{L}{2}\,(\cos \mathrm{I1} + \cos \mathrm{I2})$$

$$\Delta\mathrm{N} = \mathrm{F}\,\frac{L}{2}\,(\sin \mathrm{I1}\ \cos \mathrm{A1} + \sin \mathrm{I2}\ \cos \mathrm{A2})$$

$$\Delta\mathrm{E} = \mathrm{F}\,\frac{L}{2}\,(\sin \mathrm{I1} \sin \mathrm{A1} + \sin \mathrm{I2} \sin \mathrm{A2})$$

Where Φ, I, L, and A are the dogleg angle, inclination, measured depth between the survey points, and azimuth respectively. The numbers 1 and 2 represent the previous and next station points to be calculated. Note: the dogleg angle is obtained as presented in section 3.4.

To study the effectiveness of each method, different studies were conducted using different methods. Moreover, other studies used mathematical expressions combined with surveying methods to obtain 3D well trajectories. These different studies will be discussed in this chapter.

3.9. Bezier Curves

A Bezier curve is a parametric curve that is used to describe 2D and 3D trajectories (Sampaio Jr, 2016). It was first presented by Pierre Bezier in 1962 to be used in the automotive industry (Joshi & Samuel, 2017).

A second-order Bezier curve is defined as the following:

$$B(U) = S(1-U)^2 + 2(1-U)UCs + U^2E$$

Where S, E, and Cs are the starting point, end point, and control point of the curve. In addition, U is a dimensionless parameter that lies in the interval [0, 1] (Sampaio Jr, 2016). By replacing U in the equation above, the coordinates of all the points along the trajectory can be obtained. For example, when U=0 is replaced in the equation above, B (U=0) =S which is the coordinates of the starting point. Similarly, the coordinates of the end point is obtained when replacing U=1.

The impact of changing the control point or attractor Cs is shown in the figure below:

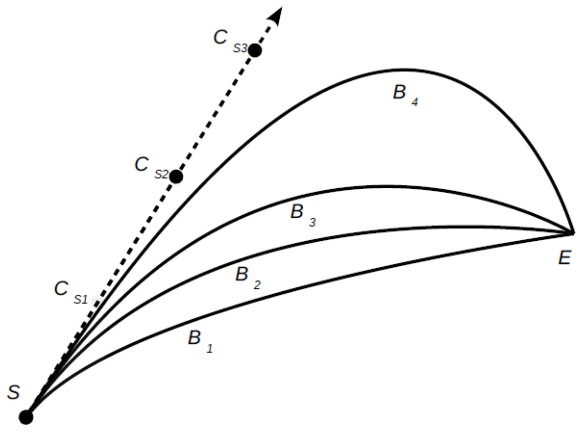


Figure 11 Control points.

Based on figure 11, any change in Cs (Cs_1, Cs_2 and Cs_3) will lead to a change in the shape of the Bezier curve (B_1, B_2 and B_3). However, based on the same figure, it can be seen that the different Bezier curves are tangent to the line that starts at S (Sampaio Jr, 2016). It is important to note that, both points S and E are fixed points that do not change when using this type of function.

A third order Bezier curve is defined as follows:

$$B(U) = S(1-U)^3 + 3(1-U)^2UCs + 3(1-U)U^2Ce + U^3E$$

Where C_E is the second control point related to the ending part of the curve (Joshi & Samuel, 2017). Similarly as to the control point Cs, changing C_E will lead to a change in the shape of the Bezier curve as seen in the figure below:

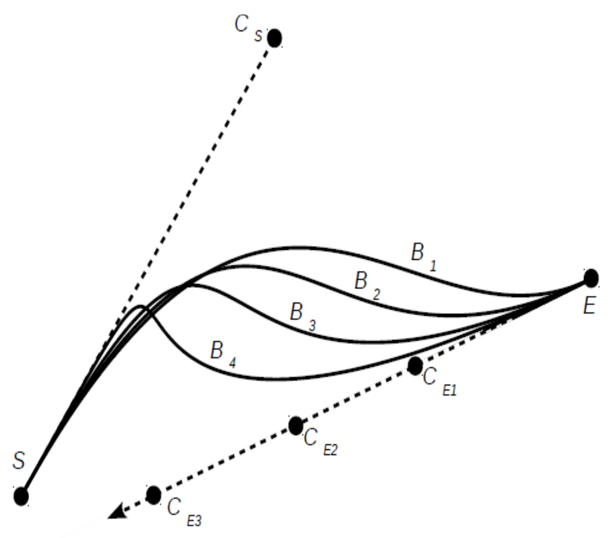


Figure 12 Control points.

According to figure 12, a change in the control point C_E (C_{E1}, C_{E2} and C_{E3}) will lead to a change in the shape of the Bezier curve (B_1, B_2 and B_3). In addition, it is evident that Bezier curves are tangent to the line that joins the end point E to the control point C_E. Based on Sampaio Jr (2016), the main difference between a 2D and 3D Bezier curve is that in the case of the latter, the lines containing S and Cs or E and C_E may not necessarily be straight lines.

The information provided will dictate the type of trajectory used when implementing Bezier curves for well path generation. These types are summarized below. ***<u>Note</u>***: The coordinates of S and E are always given independent of the type of trajectory.

- Free-end trajectory: In this category, the inclination and azimuth at the target are not given. Therefore, only the inclination and azimuth at the start point are known. In addition, the only control point present in this trajectory is Cs (Sampaio Jr, 2016).

- Set-end trajectory: In this category, the inclination and azimuth at both the start point and target are given. In addition, both control points Cs and C_E are present in the well path (Sampaio Jr, 2016).
- Set-Free and Free-set trajectory: In this category, the inclination or azimuth is not given at the target (Sampaio Jr, 2016).

The tangent vectors at S and E (ts and t_E) are calculated using the following equations:

$$ts = [\sin Is \, . \cos As \, , \sin Is \, . \sin As \, , \cos Is]$$

$$t_E = [\sin I_E \, . \cos A_E \, , \sin I_E \, . \sin A_E \, , \cos I_E]$$

Where Is, As, I_E and A_E are the given inclination and azimuth at both the start point and end point respectively (Sampaio Jr, 2016).

Once ts and t_E are calculated, the control point Cs and C_E can then be calculated using the following equations:

$$Cs = S + ds \, ts$$

$$C_E = E + d_E \, t_E$$

<u>Note</u>: If the trajectory is free-end, then only Cs is calculated. If the trajectory is set-end, then both are needed to obtain the trajectory.

Where S and E are the given coordinates of the start point and target point. In addition, ds and d_E are two arbitrary scalar parameters that provide two degrees of freedom when using Bezier curves (Sampaio Jr, 2016). These parameters may be changed to obtain the best fit depending on the given scenario.

The following equation is used to calculate the unit tangent vector for points that lie in between the start point and end point or when U>0 and U<1:

$$t = \frac{B(U)^1}{\sqrt{B(U)^1 . B(U)^1}}$$

Where $\mathrm{B(U)}^1$ is dependent on the type of the trajectory. When the trajectory is free-end, $\mathrm{B(U)}^1$ is simply the derivative of the 2D Bezier function with respect to u. It is represented as follows (Sampaio Jr, 2016):

$$B(U)^1 = -2(1-U)S + 2(1-2U)Cs + 2UE$$

When the trajectory type is a set-end trajectory, then $\mathrm{B(U)}^1$is the derivative of the 3D Bezier function with respect to u. It is represented as follows (Sampaio Jr, 2016):

$$B(U)^1 = 3(1-U)^2S + 3(1-U)(1-3U)Cs + 3(2-3U)UCe + 3U^2E$$

Once the unit tangent vector is obtained, the coordinates of the calculated tangent vector are used to obtain the inclination and azimuth using the following equations (Sampaio Jr, 2016):

$$I = \cos^{-1} tz$$

$$A = \tan^{-1} \frac{ty}{tx}$$

Note: The inclination and azimuth are used to check for dogleg severity constraints.

To calculate the trajectory length needed to intersect the target, the following equation is used (Sampaio Jr, 2016):

$$MD = L(Ui) = \sum_{0}^{i} \sqrt{(\Delta xi)^2 + (\Delta yi)^2 + (\Delta zi)^2}$$

Where Δxi, Δyi, and Δzi is simply the difference between the coordinates of two consecutive points. Since this expression is integrated numerically, then the length obtained is always slightly smaller than the actual trajectory length (Sampaio Jr, 2016).

3.10. Different Methods Used to Obtain Real (3D) Trajectories.

Based on McMillian (1980), the usage of the radius of curvature method to calculate the trajectory of a well is a practical method to obtain a well path. In addition, acoording to McMillian (1980), this method may be applied on all directional wells. To account for the simultanous change in azimuth and inclination, and to simplify the problem presented, the usage of a 3x3 matrix was required. This matrix was used to convert the well path from 3D trajectory to 2D (McMillian, 1980). Once the trajectory was calculated in 2D, the solution would then be transferred back into a 3D trajectory by the usage of the inverse matrix (McMillian, 1980) .

Studies to calculate the 3D trajectory of a well using mathmtical expressions were also conducted. Scholes (1983) devloped an algorithm that would obtain 3D trajectories by the usage of cubic polynomials. In this study, the coordinates of the points along the path were determined by identifying the coeffecients of the polynomials using the information provided regarding the given end points and their respective inclination and azimuth.

McClendon and Andres (1985) developed an algorithm that plots 3D well paths by the usage of the catenary model. Based on the research conducted, it was shown that using this model provided concrete advantages over other methods used.

According to Bourgoyne et al. (1991), the tangential method was the most used method in the industry. However, due to the large errors in this method, other methods gained importance. To determine the most accurate surveying method, (Bourgoyne et al., 1991) compared the results obtained from the different surveying methods after using data obtained from “test holes”. The results, which were presented in a table, indicated that the difference between the minimum curvature method, radius of curvature method, and balanced tangentail method were negligible (Bourgoyne et al., 1991). Therefore, any of the three mentioned methods could be used. However, based on Bourgoyne et al. (1991), due to the enhancement in programmable calculators, the most common method used in today's industry is the minimum curvature method.

Guo et al. (1992) also implemented the usage of mathematical models to plan well trajectories. In this study, the method implemented was the usage of the constant curvature method.

On the other hand, Konstantakopoulos and Stamataki (1996) focused on the ability to controll the trajectory of a well while drilling. To monitor the change in the position of the well, the radius of curvature method was implemented.

Sawaryn and Thorogood (2003) aimed at providing unified nomenclatures and mathematical conventions for the implementation of the minimum curvature method. In addition, the different limitations when using this algorithm were also covered and discussed in this study.

Sampaio (2006) further developed the usage of cubic polynomials to calculate 3D trajectories. In this paper, different end conditions were introduced depending on the information provided. The first end condition is the free end condition. In this condition, both the azimuth and inclination of the target are uknown. On the other hand, the set end condition is when both inclination and azimuth of the target are provided. The third end condition is the set inclination and free azimuth. Thus, in this scenario, the inclination is specified while the azimuth is unkown. Lastly, the fourth given condition is the set azimuth and free inclination. In this end condition the azimuth is known while the inclination is uknown. However, using cubic polynomials meant the presence of a limitation that does not provide any degree of freedom when calculating trajectories (Sampaio, 2006).

To solve this issue, Sampaio (2007) presented the usage of spline-in-tension functions to plot 3D trajecotries. These functions are simply continuous functions obtained as solutions by solving differential equations. By using this method, Sampaio (2007) was able to provide a parameter called tension. This parameter allowed the algorithm to have a certain degree of freedom when calculating trajectories (Sampaio, 2007).

To reduce the time required to solve the different complex equations used for well trajectory design, Amorin and Broni-Bediako (2010) designed an excel spreadsheet that implemented the different surveying methods such as the radius of curvature method and constant curvature method. Also, this excel spreadsheet provides both vertical and horizontal views of the well trajectory.

In the case study conducted by Amorin and Bediako (2010), all five different surveying techniques were tested. In the following article, the author devloped an “excel spreadsheet” to perfrom the calculations required (Amorin & Bediako, 2010). According to Amorin and Bediako (2010), the data used were obtained from two different studies found in the literature. This data was then used as separate inputs in the spreadsheet and the results of each case was obtained. These results were then compared to the results of the respective studies for verification. Therefore, after comparing the calculated outputs with the outputs found in the literature, Amorin and Bediako (2010) deduced that the difference between the average angle, balanced tangential, radius of curvature, and minimum curvature surveying methods was very small. Therefore, all four different methods may be used. On the other hand, the tangential method was considered to be inaccurate. However, Amorin and Bediako (2010) concluded that the most accurate method between the four methods stated was the minimum curvature method. This was supported by the fact that the results obtained by using this method contained the smallest error.

More studies on the radius of curvature method were conducted. Based on Krishnan and Kulkarni (2016), the radius of curvature method was used to generate a well path for the build and hold well profile. Once the tajectory was generated, it was compared to the output from the "COMPASS" software which is commonly used in the industry for well planning (Krishnan & Kulkarni, 2016). The results obtained indicated that similarity between both outputs exist (Krishnan & Kulkarni, 2016).

According to Sampaio (2016), the transformation of Bezier curves into 3D may generate a practical method to calculate well trajectories. Moreover, Sampaio (2016) also stated that this mathematical model performs better and is more flexible when compared to other algorithms that use cubic polynomials or spline-in-tension functions to define 3D trajectories. To support this claim, Sampaio (2016) stated that the trajectory of a well is obtained by using a single expression rather than working with three separate coordinate functions. In addition, two arbitrary scalar parameters were added that increased the flexibility of the algorithm.

Another study that used mathematical modeling for the calculation of 3D well trajectories was conducted by Okpozo et al. (2016). Based on Okpozo et al. (2016), this mathematical model would ensure not to surpass the maximum dogleg allowed in the scenario being tested. This model needed the usage of a "curved horizontal displacement" rather than the standard straight displacement (Okpozo et al., 2016). Therefore, the authors first derived the necessary equations to fit in their mathematical model. Once the derivations were established, the authors then developed a software known as "WELLTIT" (Okpozo et al., 2016). This software depends on the radius of curvature surveying method to calcualte well trajectories. Therefore, the vailidity of this mathematical model was tested by the usage of the developed software (Okpozo et al., 2016). The results obtained indicated that the usage of the mathematical model proved to be working “effectively” (Okpozo et al., 2016).

Joshi and Samuel (2017) combined the usage of both surveying methods and mathematical expressions to correct any departure from a planned path. In this research paper, the authors proposed a “closed loop system” that automates the process of path correction and determines the best path based on the current location of the well (Joshi & Samuel, 2017). This study was based on different surveying methods such as the balanced tangential and minimum curvature method. In addition, the authors used different mathematical functions such as the Bezier curves and cubic polynomials (Joshi & Samuel, 2017).

On the other hand, Wang et al. (2019) used the “vector algebra method” with the minimum cruvature method to design a 3D well trajectory. In this study, the shape of the target was taken into consideration to ensure that the well is following the correct path (Wang et al., 2019). Therefore, different mathemtaical expression were derived based on the target’s shape. The purpose of this study was to explain the derivtions related to the formulation of the expressions. Then, Wang et al. (2019) implemented the newly derived expressions to confirm that the model would function. Based on the results, Wang et al. (2019) deduced that the vector algebra method is suitable for the usage in determining well trajectories.

3.11. The Method Selected For this Study.

For this study, Bezier curves will be used as the function needed to calculate 3D trajectories. This can be supported by the fact that the usage of such functions has provided added flexibility by the presence of two control points. The presence of control points in such functions allows the user to modify the curvature to fit certain requirements. This allows the planned trajectory to intersect horizontal well targets in a smooth and curved manner. This is why the Bezier model provides two degrees of freedom. While both the spline-in-tension model and cubic polynomial model provide one and zero degrees of freedom respectively.

In addition to flexibility, the Bezier curve model is capable of obtaining the exact position of points by the usage of a single expression. However, for the catenary model, multiple expressions are needed for the same purpose. On the other hand, for determining the coordinates of points along the well path, the constant curvature method requires the solving of integrals using numerical approximations which may lead to unwanted errors.

As a result, the Bezier model is capable of calculating well trajectories in a simple, accurate, and flexible manner. This explains why it was selected as the model of choice for well path design in this thesis study.

CHAPTER 4

METHODOLOGY

This section will be divided into two main parts. The first part will discuss the different components that were implemented in the algorithm.While as, the second part will be related to the input parameters, outputs, and the algorithm structure.
To successfully design the 3D well trajectory required, an algorithm was generated in Python. This algorithm implements the usage of different components that were presented in this paper.

4.1. Curve Used to Generate 3D Well Trajectories.

As stated in section 3.9, Bezier Curves will be used to model 3D well trajectories. These curves were selected due to the added flexibility that is provided by the presence of the two control points. These control points ensure that the planned trajectory is capable of intersecting horizontal well targets efficiently and smoothly. Therefore, the equations present in section 3.9 were implemented to generate the needed algorithm.

4.2. Azimuth.

The azimuth used in the algorithm being discussed accounts for changes in three-dimensional space. As explained in section 3.3, the azimuth is the angle formed with the horizontal plane. Based on trigonometric functions, the inverse tangent is used to calculate the azimuth of a line that joins two points. This is done by applying the formula below:

$$Azimuth = \tan^{-1}\left(\frac{dy}{dx}\right)$$

Where dy and dx are simply the difference in the y and x coordinates of the two points being studied.

However, the inverse function “arctan” used in Python will provide a solution in the range of [-90, 90] degrees. Thus, it is evident that this inverse function cannot account for the azimuth on all four quadrants. To solve this issue, the inverse function “arctan2” was implemented. This inverse function provides the solution in the range of [-180,180] degrees. The answer provided by using this function is in radians and is measured anticlockwise from the positive x-axis. The equation of arctan2 is the following:

$$Azimuth = arctan2\ (dy, dx)$$

It is important to note that, based on section 3.3, the azimuth is always calculated in the clockwise direction with respect to the north or positive y-axis. To be consistent with this definition, some modifications were implemented to convert the obtained angle. As a result, the final equation that calculates the azimuth is the following:

$$Azimuth = -arctan2\ (dy, dx) + \frac{\pi}{2}$$

4.3. Inclination.

As described in section 3.2, the inclination is the angle formed with the vertical plane. For this study, the inclination was calculated using the equations described in section 3.9.

4.4. Measured Depth.

For this study, the measured depth was calculated using the equation described in section 3.9.

4.5. Dogleg Severity.

As stated in section 3.4, dogleg severity is a function of the dogleg angle. Thus, the equations presented in this section were implemented. The maximum allowable DLS was chosen based on the range defined by Mitchel (1995), who stated that the DLS of a well may range from 3 to 10 depending on the well depth. Therefore, all well trajectories planned by the algorithm must not exceed a maximum DLS limit which was set at 7.

4.6. Requirements of the Algorithm.

The input parameters needed for the algorithm to function are the following:

1- Surface coordinates or drilling center.

2- Target coordinates.

3- DLS maximum limit.

4- Kick-Off point.

5- Arbitrary parameter to reach Cs.

6- Parametric t.

4.6.1. Surface coordinates or drilling center.

The surface coordinates are specified by the OGED team as will be seen in the testing cases. This point is also known as the drilling center.

4.6.2. Target coordinates.

The target coordinates are specified by the OGED team as will be seen in the testing cases. Normally, the target is a horizontal or vertical well that consists of multiple points.

4.6.3. DLS maximum limit.

The DLS maximum limit is set by the user. For the cases tested in this paper, a DLS limit of 7 was set. It is important to note that, the higher the DLS, the lower the needed measured depth needed to reach the target.

4.6.4. Kick-Off point or KOP.

In some drilling scenarios, the kick-off point may be provided by the geologist based on the formation. In other cases, the directional engineer is responsible for determining its position (Inglis, 1987). In this study, the kick-off point is a decision variable in the optimization problem. Thus, it will be specified by the optimizer.

4.6.5. Arbitrary parameter to reach Cs.

As described in section 3.9, the control parameter Cs is calculated using the following equation:

$$Cs = S + ds\ ts$$

Based on this study, the starting point or “S” is the point that lies vertically below the drilling center at a depth equal to KOP. In addition, ts can also be calculated using the equation discussed in section 3.9. Thus, the control point Cs can be obtained after setting a value for the arbitrary parameter to reach Cs designated by ds. In this study, ds is a decision variable in the optimization problem. Thus, it will be specified by the optimizer.

4.6.6. Parametric t.

To calculate the second control point Ce, a modification was implemented to ensure that the planned well trajectory intersects the target smoothly. Based on the definition of a Bezier curve, the trajectory will always be tangent to the line that joins the control point and the initial or end point (Sampaio Jr, 2016). To apply this modification to the algorithm, the following steps were used:

1- Calculate the direction vector of the horizontal well target between the target point and the point that directly follows it.

2- Calculate the parametric equation of the line between these two points.

3- Specify the value of the parametric t to calculate the control point Ce.

By using these steps, the control point Ce will always be collinear with the target. This will allow the calculated trajectory to intersect the target smoothly. In this study, parametric t is a decision variable in the optimization problem. Thus, it will be specified by the optimizer.

4.7. Algorithm Structure.

As described throughout this paper, an algorithm was designed and implemented in Python for well path calculation. This algorithm follows the following structure:

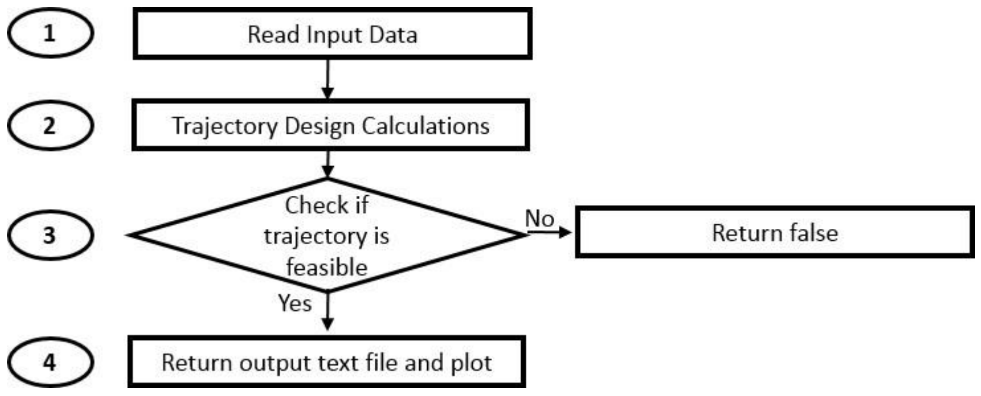

1
Read Input Data
2
Trajectory Design Calculations
3
Check if trajectory is feasible
No
Return false
Yes
4
Return output text file and plot

Figure 13 Algorithm structure (Level 1)

Based on Figure 13, the algorithm in level 1 is divided into the following sections:

1- Read Input Data.

2- Trajectory Design Calculations.

3- Check if Trajectory is feasible.

4- Return Output Text File and Plot.

4.7.1. Read Input Data.

For the algorithm to function, the OGED team will provide an input text file that will contain the well that needs to be intersected also known as the targeted well. This text file will also contain the Surface coordinates or drilling center as explained in section 4.6.1. Thus, for each well that needs to be intersected, a text file will be provided. Hence, once this text file has been introduced, the algorithm will identify the surface point and store it in the variable “drilling_center”. In addition, it will store the targeted well in the variable “well_array”.

4.7.2. Trajectory Design Calculations.

This section is further divided into the following parts (level 2):

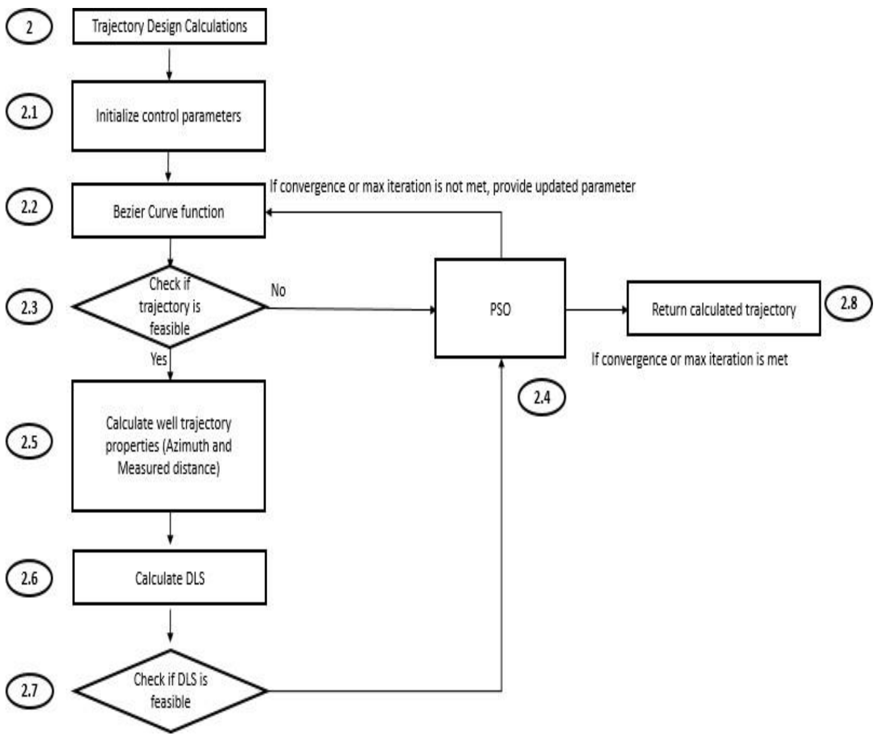

2
Trajectory Design Calculations
2.1
Initialize control parameters
2.2
Bezier Curve function
If convergence or max iteration is not met, provide updated parameter
2.3
Check if trajectory is feasible
No
Yes
PSO
2.4
Return calculated trajectory
2.8
If convergence or max iteration is met
2.5
Calculate well trajectory properties (Azimuth and Measured distance)
2.6
Calculate DLS
2.7
Check if DLS is feasible

Figure 14 Trajectory design calculations structure (Level 2)

Based on figure 14, the trajectory design calculations section in level 2 is divided into the following parts:

1- Initialize Control Parameters.

2- Bezier Curve Functions.

3- Check if Trajectory is feasible.

4- Particle Swarm Optimization (PSO).

5- Calculate Well Trajectory Properties.

6- Calculate DLS.

7- Check if DLS is feasible.

8- Return Calculated Trajectory.

4.7.2.1. Initialize Control Parameters.

The control parameters of this algorithm are the kick-off point, arbitrary parameter to reach Cs, and parametric t. Each control parameter is discussed in detail in section 4.6. In this section, the three parameters are initially set with a random number.

4.7.2.2. Bezier Curve Function.

This function is divided into different sections as shown in the figure below (level 3):

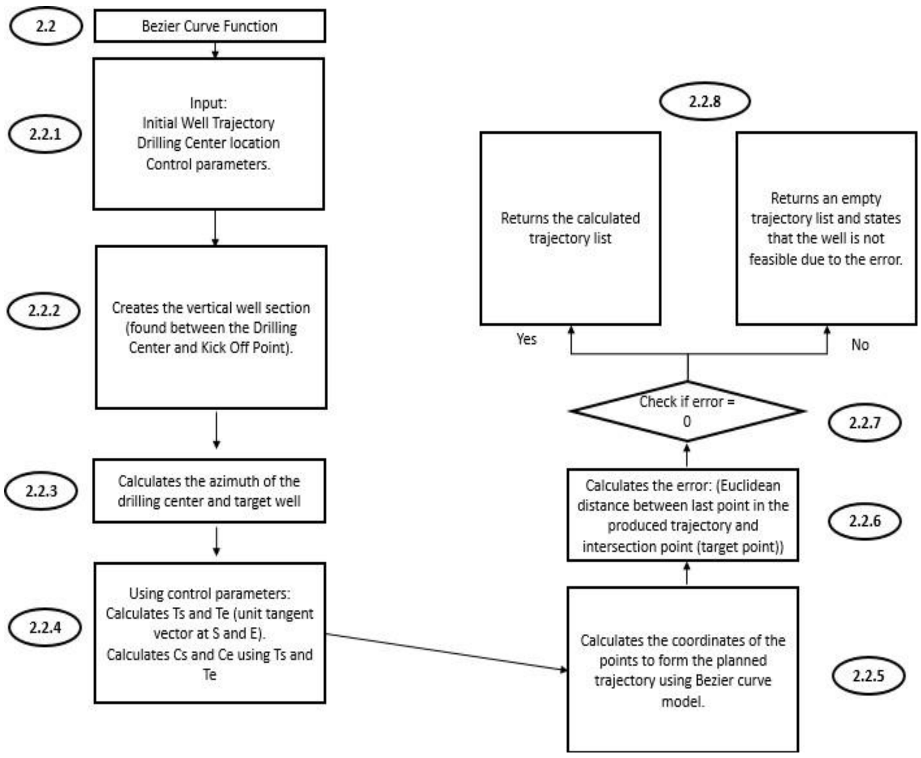

2.2
Bezier Curve Function
2.2.1
Input: Initial Well Trajectory Drilling Center location Control parameters.
2.2.2
Creates the vertical well section (found between the Drilling Center and Kick Off Point).
2.2.3
Calculates the azimuth of the drilling center and target well
2.2.4
Using control parameters: Calculates Ts and Te (unit tangent vector at S and E). Calculates Cs and Ce using Ts and Te
2.2.5
Calculates the coordinates of the points to form the planned trajectory using Bezier curve model.
2.2.6
Calculates the error: (Euclidean distance between last point in the produced trajectory and intersection point (target point))
2.2.7
Check if error = 0
Yes
No
2.2.8
Returns the calculated trajectory list
Returns an empty trajectory list and states that the well is not feasible due to the error.

Figure 15 Bezier curve function structure (Level 3)

Based on figure 15, the Bezier Curve function section in level 3 is divided into the following parts:

1- Input Initial Well Data.
2- Creates the Vertical Well Section.
3- Calculates the Azimuth of the Drilling Center and Target Well.
4- Uses Control Parameters.
5- Calculates the Coordinates of the Points to form the Planned Trajectory using Bezier Curve Model.
6- Calculates the Error.
7- Checks the Error.
8- Return Trajectory List.

4.7.2.2.1. Input Initial Well Data.

For this part, the algorithm will simply receive the inputs that are required for normal functioning. Thus, it will receive the input parameters, drilling center location, and initial well trajectory.

4.7.2.2.2. Create the Vertical Well Section.

The vertical segment of the well is generated in this section. Therefore, a loop is implemented to generate all the points that lie in this section. This loop will simply start at the coordinates of the drilling center. Then, by only changing the z-axis coordinates, the loop will add an increment of 30 meters to reach the starting point or “S” which has a z-axis value equal to the kick-off point. All the points generated in this section will be added to the list designated by ”trajectory_list”. In addition, in this section, the algorithm will specify the point of intersection or target point. This point is assigned based on the minimum

Euclidean distance calculated from the starting point “S” to the well extremities (both well heel and well toe). Thus, this specified point will be the point that needs to be intersected by the calculated Bezier curve trajectory.

4.7.2.2.3. Calculates the Azimuth of the Drilling Center and Target Well.

In this section, the azimuth of both the drilling center and well target will be calculated using the method described in section 4.2.

4.7.2.2.4. Use Control Parameters.

In this part of the function, the tangent vectors at the starting point “S” and end point “E” are calculated using the equations presented in section 3.9. In addition, based on the given input parameters, control points Cs and Ce are also calculated using the equations described in sections 4.6.5 and 4.6.6.

4.7.2.2.5. Calculates the Coordinates of the Points to form the Planned Trajectory using Bezier Curve model.

In this section, all the points that lie in this curved segment are generated using the Bezier curve equations presented in section 3.9. These points are added to the “trajectory_list”. In addition, the inclination of all the points is calculated in this section as described in section 3.9.

4.7.2.2.6. Calculates the error.

This part of the function calculates the distance between the last point present in the “trajectory_list” and the target point. This distance is known as the error distance.

4.7.2.2.7. Checks the error.

In this section, the error distance is checked. If this distance is greater than 0.01 or less than 0, then the calculated well is not feasible and the “trajectory_list” is emptied.

4.7.2.2.8. Return Trajectory List.

This part of the Bezier curve function returns the obtained “trajectory_list”.

4.7.2.3. Check if Trajectory is feasible.

In this section, the length of the “trajectory_list” is checked. If the length is not empty or greater than 0, the algorithm will continue with the different functions. However, if the length of the “trajectory_list” is empty or 0, then the optimizer “PSO” will be introduced.

4.7.2.4. Particle Swarm Optimization (PSO).

According to Ahmad et al. (2019), the particle swarm optimization algorithm was designed based on the behavior of schools of fish or flocks of birds. Fish and birds normally travel based on the position and velocity that is provided by the group. To convert this into an optimization problem, the objective function solution is represented by a particle (Ahmad et al., 2019). Also, the group of particles is represented by a swarm (Ahmad et al., 2019). From the previous iteration, a velocity parameter is used to affect the position of every particle (Ahmad et al., 2019). This velocity parameter is modified based on a mathematical function that uses the following three pieces of information. The first piece of information used is the particle’s previous velocity. The second piece of information used is the distance to where the particle achieved the local best. Finally, the last piece of information used is the distance to where the particle achieved the global best (Ahmad et al., 2019). It is important to state that, every particle in this algorithm will store the local best position it achieved. In addition, the algorithm itself will store the global best position achieved between all the particles known as the swarm global best (Ahmad et al., 2019). The equations presented below are used to calculate the position and velocity of each particle (Ahmad et al., 2019).

$$Xi,j(K+1) = Xi,j(K) + Vi,j(K+1)$$

$$Vi,j(K+1) = w * Vi,j(K) + Cp * r1(Pbest(i,j) - Xi,j(K)) + Cg * r2(gbest(i,j) - Xi,j(K))$$

Where i, j, and k refer to the particle, optimization variable, and current iteration respectively (Ahmad et al., 2019). Also, in the velocity equation, w, cp, cg, r1, and r2 refer to the inertia weight, cognitive weight, social weight, and independent uniform random variables respectively (Ahmad et al., 2019).

The inertia weight, cognitive weight, and social weight provide the influence of how the swarm moves towards the particle velocity, particle local best, and swarm global best (Ahmad et al., 2019). To prevent the algorithm from falling into a local optimum, the random variables r1 and r2 were introduced. These variables range between 0 and 1 and allow the optimization problem to become stochastic (Ahmad et al., 2019).

This algorithm usually requires the presence of a convergence criteria. This is the criteria that specifies when the algorithm converges. Thus, the value of the difference in the objective function between two particles of the same iteration will be calculated. The optimization problem converges when this difference is equal to or less than the specified convergence criteria. Another requirement for the particle swarm optimization algorithm is, the specification of the number of particles present in the optimization problem. Finally, the maximum number of iterations must also be defined. This will prevent the algorithm from falling into an infinite loop in case it gets stuck in a local minimum solution.

The purpose of this study is to provide the shortest distance from the starting point to reach the target point while satisfying the DLS constraint. Thus, it is obvious that the PSO optimizer implemented needs to minimize the measured depth to reach the target point (which is the objective function of the optimization problem). This is done by providing an input to the Bezier curve function with the three optimal parameters that were initially set at a random number as discussed in section 4.7.2.1 (level 2).

It is important to note that, the PSO optimizer will keep on providing different input parameters to the Bezier Curve function until it meets the maximum number of iterations or converges. Once it meets one of these two conditions, it will terminate and send the obtained results to section 4.7.2.8 (level 2).

4.7.2.5. Calculate Well Trajectory Properties.

In this section, the azimuth of all the points in “trajectory_list” is calculated as described in section 4.2. In addition, the measured depth to reach the intersection point is also calculated as described in section 4.4.

4.7.2.6. Calculate DLS.

In this section, the DLS of all the points in “trajectory_list” is calculated as described in section 3.4.

4.7.2.7. Check if DLS is feasible.

In this section, every DLS obtained in the previous section is compared to the DLS limit that is set at 7. If one DLS element obtained is larger than the DLS limit then the whole well is not feasible and the "trajectory_list" is emptied. The output of this section is sent back to the PSO optimizer.

4.7.2.8. Return Calculated Trajectory.

In this section, the"trajectory_list" is returned as the output. This occurs once the optimizer has either converged or attained the maximum number of iterations allowed.

4.7.3. Check if trajectory is feasible.

In this section, the length of "trajectory_list" will be checked for the final time. If the trajectory is empty, then this section will return false. Otherwise, it will send the list to the final section below.

4.7.4. Return output text file and plot.

In this section, the output text file that contains the trajectory of the well, and all other properties is created. In addition, a plot in python is generated to show how the calculated well (represented as blue in the plot) will intersect the target (represented by red in the plot).

CHAPTER 5

OPTIMIZATION AND TESTING

As seen in chapter 4, the Bezier curve algorithm was designed and optimized using the Particle Swarm Optimization algorithm. This selection was done to provide consistency with using the same optimization algorithm as used by the OGED team in well placement and facility optimization. For further testing, different optimizers were linked to the bezier curve algorithm and tested. The results obtained will be shown in this chapter.

5.1. Timeline of the Different Studies Done for Well Trajectory Optimization.

Multiple studies exist in the literature that tackle the optimization of the design of well trajectories. These studies will be presented briefly in the table below.

Table 2 Timeline of different studies conducted on well path optimization.

Author	Description of study
Helmy et al., 1998	In this research paper, the authors implemented the usage of non-linear optimization based on the sequential unconstrained minimization technique (SUMT) to minimize drilling depth and reduce drilling costs of an S-type well profile (Helmy et al., 1998).

Shokir et al., 2004	In this research paper, the authors implemented the usage of the Genetic Algorithm (GA) to optimize horizontal and directional well parameters using the radius of curvature method (Shokir et al., 2004).
Schulze-Riegert et. Al., 2011	The authors of this study implement the usage of the stochastic optimizer (Covariance Matrix Adaption Evolution Strategy) (Schulze-Riegert et. Al., 2011).
Arabjamaloei et al., 2011	In this research study, the authors used artificial neural network to optimize well trajectories (Arabjamaloei et al., 2011).
Seyyedattar et al., 2012	The authors of this study optimize the process of locating the well head and well path design via one step using the well optimizer module of Petrel which is the curvature method. (Seyyedattar et al., 2012).
Atashnezhad et al., 2014.	The usage of the particle swarm optimization (PSO) to optimize horizontal and directional well parameters using the radius of curvature method (Atashnezhad et al., 2014).
Guria et al., 2014.	The authors of this study use the elitist non-dominated sorting genetic algorithm for multi-objective optimization (Guria et al., 2014).

Mansouri et al., 2015.	The usage of a multi-objective genetic algorithm to minimize MD and frictional torque based on the radius of curvature method (Mansouri et al., 2015).
Tianshou et al., 2015.	The usage of breakout width model and Mogi–Coulomb criterion to optimize both well trajectory and wellbore stability (Tianshou et al., 2015).
Wood 2016.	The Hybrid cuckoo search optimization algorithm was used to optimize the trajectories of wells using the radius of curvature method (Wood, 2016).
Wood, 2016.	The usage of the Hybrid bat flight optimization algorithm to optimize the trajectories of wells using the radius of curvature method (Wood, 2016).
Salman et al., 2017.	The authors of this study conducted optimization by combining firefly algorithm with hybrid implementations of mutation and annealing to well path produced by spline curves (Salman et al., 2017).
Sha & Pan, 2018.	The usage of Fibonacci sequence-based self-adjustment quantum genetic algorithm to optimize trajectories based on the radius of curvature method (Sha & Pan, 2018).
Halafawi & Avram, 2018.	Optimization by selecting the KOP, horizontal turn trajectory, vertical turn determination mud weight,

	and horizontal profile. After optimization, the 3D trajectory was produced using the minimum curvature method (Halafawi & Avram, 2018).
Wi´sniowski et al., 2020.	The optimization using two methods (GA and State Space Search) of trajectories produced by the classical sectional trajectory that was modified to model the catenary trajectory (Wi´sniowski et al., 2020).
Biswas et al., 2020.	The optimization of multiple objective functions using the hyena optimizer (Biswas et al., 2020).

5.2. A Brief Description of the Different Optimization Algorithms tested.

A different combination of both gradient and non-gradient based algorithms was tested. These algorithms are the following:

1- Particle Swarm Optimization or PSO.

2- Differential Evolution or DE.

3- Genetic Algorithm or GA.

4- Dual Annealing or DA.

5- Sequential Least Square Programming or SLSQP.

5.2.1. Particle Swarm Optimization.

A detailed description of this optimization algorithm was presented in section 4.7.2.4.

5.2.2. Differential Evolution.

Similarly, to PSO, the DE algorithm requires the presence of particles. These particles are considered as solutions to the optimization problem. In this algorithm, the search space is covered by the generation of vector populations (Price et al., 2005). Once the search space has been populated, the algorithm will select two vectors randomly and will calculate the difference between these two vectors (Price et al., 2005). Then, the difference is added to a third randomly selected vector and multiplied by a mutant factor to form the mutant vector (Price et al., 2005). The building of a trial vector is done by the selection of parameters from two different mutant vectors. This selection is controlled by the probability of crossover which will dictate the fraction of the different values copied from the mutant vector (Price et al., 2005). Once the trial vector has been built, it will be compared to a target population vector (Price et al., 2005). The vector that has the lowest fitness (objective function value) will be used to form the population of the next iteration. This continues until a certain limit is reached. As an input, this algorithm requires the user to set the number of particles and a maximum number of iterations. (This will be discussed in the next sections of this chapter).

5.2.3. Genetic Algorithm.

Similar to PSO and DE, this algorithm is a population-based algorithm (Mirjalili, 2018). It is based on the survival of the fittest creature (Mirjalili, 2018). Each particle in this optimization algorithm represents a chromosome (Mirjalili, 2018). During each iteration or generation, the fitness of each particle is reviewed, and the particles with the lowest fitness are selected to form the new generation. To prevent the GA from falling into local optima, the selection of the fittest particles is done stochastically (Mirjalili, 2018). This will allow the algorithm to choose poor solutions as well, and thus prevent the algorithm from falling into a trap (Mirjalili, 2018). The crossover between particles and the mutation or change in the gene of the chromosomes also provide the algorithm with diversity (Mirjalili, 2018). This continues until a certain limit is reached. As an input, this algorithm requires the user to set the number of particles and a maximum number of iterations. (This will be discussed in the next sections of this chapter).

5.2.4. Dual Annealing.

This algorithm is based on the annealing of a metal which is the heating and controlled cooling of the metal to reach its crystalline state (Xiang et al., 2013). Therefore, the objective function of this algorithm is simply the energy function of the annealing process (Xiang et al., 2013). This function will be cooled based on different cooling formulas to reach the global optimum (Xiang et al., 2013). Initially, the algorithm will start at a given initial artificial temperature which is normally very high. Then, the temperature will gradually decrease, and this will cause the energy function to change. In this decrease, the algorithm will be visiting and evaluating the energy function for the different regions of the search space. The visiting of the algorithm from one region to the next is based on a visiting distribution (Xiang et al., 2013). Also, during the visiting of different regions, the algorithm will accept/reject regions based on an acceptance probability which is a generalized Metropolis algorithm (Xiang et al., 2013). This continues until a certain limit is reached. As an input, this algorithm requires the user to set a maximum number of iterations. (This will be discussed in the next sections of this chapter).

5.2.5. Sequential Least Square Programming.

This algorithm is used to iteratively solve non-linear optimization problems (Kraft, 1988). Thus, the algorithm must first be provided with an initial guess "X0" (Kraft, 1988). This initial guess is updated based on the search direction multiplied by the step length. The search direction is found by having a quadratic subproblem (Kraft, 1988). This subproblem is generated by applying an approximation to the LAGRANGE function (Kraft, 1988). The SLSQP algorithm uses only first-order data to estimate the Hessian matrix of the function (Lagrange) (Kraft, 1988). In the case that the optimization problem has equality constraints, then this method will implement the usage of the Kuhn-Tucker system of conditions (Kraft, 1988). The problem continues until certain criteria has been met which may range from a maximum set of iterations to satisfying a certain threshold or error.

5.3. Results of the Different Optimization Algorithms.

As described previously, the different algorithms presented above were tested on the Bezier curve algorithm. A maximum number of 500 iterations was set to all the optimizers. All non-gradient-based and particle-based optimizers have 15 particles. The testing that will be presented below was conducted on five different wells. Each well was tested 100 times. It is important to note that, both PSO and DE were combined with SLSQP to test the effectiveness of having a mix between gradient and non-gradient optimization. Non-gradient particle-based algorithms will converge when the difference between the average particle and best particle is less than or equal to 0.05. For the non-gradient and non-particle-based algorithms, the algorithm converges when 50 iterations pass without having any improvement in the objective function. Finally, for the gradient-based algorithm, the optimizer converges when the difference in the objective function is less than 10^{-6}.

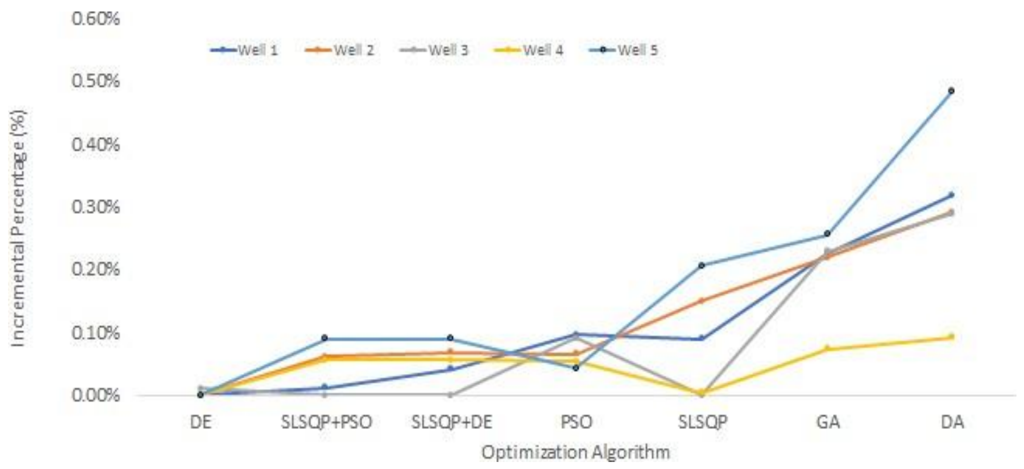


Figure 16 Results: Incremental percentage of the Measured Depth vs Optimization Algorithm.

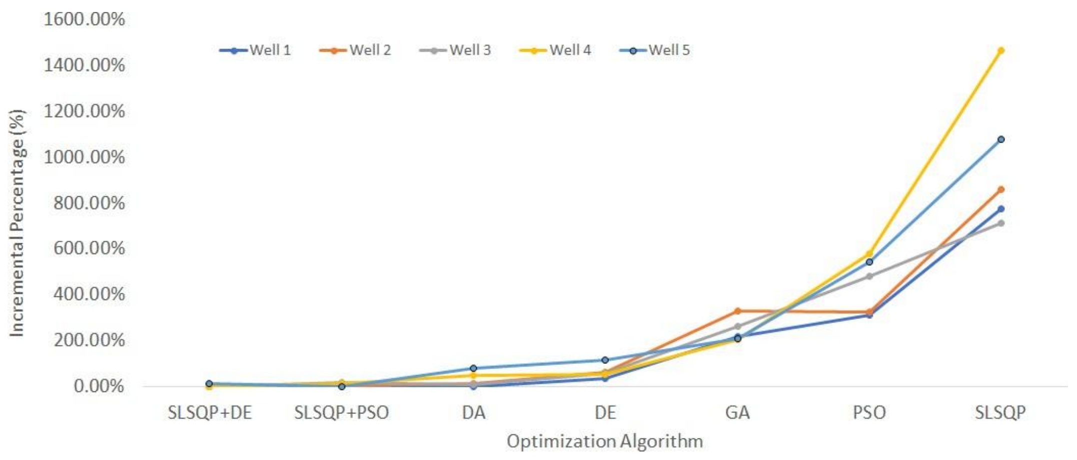


Figure 17 Results: Incremental percentage of the CPU vs Optimization Algorithm.

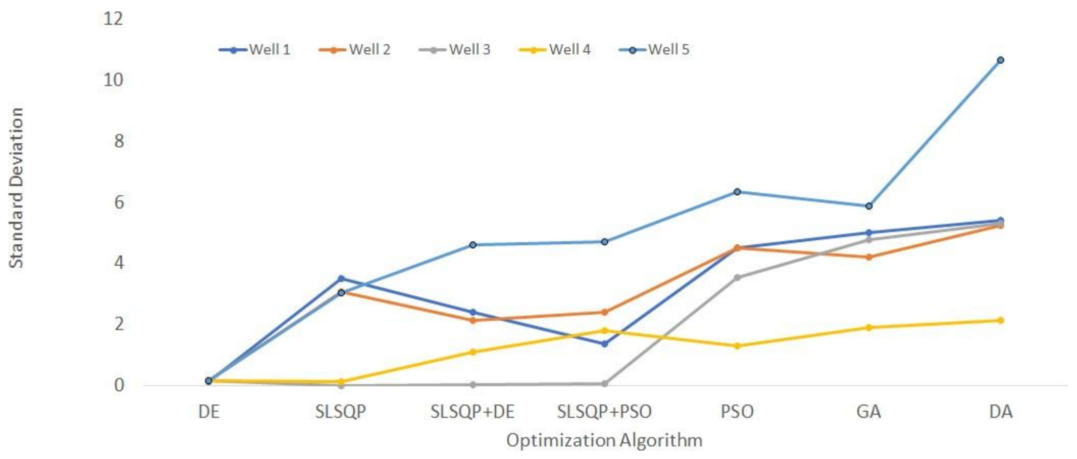


Figure 18 Results: Standard Deviation vs Optimization Algorithm.

5.4. Discussion.

Based on figures 16,17, and 18, it can be deduced that using the DE algorithm will provide the lowest average measured depth. Also, this algorithm is the most stable method between the different methods since it provided a standard deviation of almost 0. In terms of speed, combining DE with SLSQP outperformed DE. However, DE is still relatively fast with respect to the other optimizers. Thus, for optimizing well trajectory design using the Bezier curve model, DE is the best optimizer. However, for the purpose of this research study, (this was explained before), PSO will be implemented to provide consistency with the algorithm used by the OGED for optimization.

CHAPTER 6

RESULTS AND DISCUSSION

This chapter will be divided into two main parts. The first part will include the sensitivity analysis conducted to specify the values of the convergence criteria, swarm size, and maximum number of iterations needed for the PSO algorithm to produce optimal and robust solutions. The second part of this chapter will include both the results obtained and discussions after implementing the parameters found in the first part of this chapter.

6.1. Sensitivity Analysis.

Firstly, a sensitivity analysis was conducted to decide on the needed number of particles for the optimization problem to function properly. Thus, for the same wells being tested, the number of particles was changed between the following values: 15, 25, 35, and 45. The obtained measured depth and processing time for each case was then studied and compared. The results are found in the two tables below:

Table 3 Measured depth results after changing the number of particles.

Well Number:	Number of Particles: 15 Measured Depth	Number of Particles: 25 Measured Depth	Number of Particles: 35 Measured Depth	Number of Particles: 45 Measured Depth
1	2555.844 m	2554.251 m	2554.277 m	2554.242 m

2	2547.401 m	2547.426 m	2547.377 m	2547.380 m
3	2528.480 m	2529.064 m	2528.525 m	2528.468 m
4	2464.580 m	2463.633 m	2466.334 m	2463.490 m
5	2707.768 m	2707.728 m	2707.726 m	2707.710 m
6	2646.0795 m	2653.893 m	2617.318 m	2485.512 m
7	Not Feasible	Not Feasible	Not Feasible	Not Feasible
8	2917.597 m	2893.994 m	2894.015 m	2894.003 m
9	2482.192 m	2481.849 m	2481.837 m	2482.582 m
10	2628.687 m	2628.470 m	2628.411 m	2628.414 m
Average MD	2608.737 m	2606.701 m	2602.869	2587.978 m

Table 4 Processing time results after changing the number of particles.

Well Number	Number of Particles: 15 Processing time	Number of Particles: 25 Processing time	Number of Particles: 35 Processing time	Number of Particles: 45 Processing time
1	18.522 s	28.427 s	44.740 s	58.700 s
2	12.951 s	29.496 s	44.409 s	57.004 s
3	16.174 s	24.284 s	52.727 s	48.125 s
4	18.040 s	24.983 s	53.405 s	20.983 s
5	17.588 s	21.511 s	47.989 s	18.221 s
6	0.696 s	0.4713 s	1.493 s	8.346 s

7	Not Feasible	Not Feasible	Not Feasible	Not Feasible
8	5.949 s	17.529 s	38.071 s	42.541 s
9	17.548 s	22.315 s	18.481 s	54.029 s
10	14.214 s	23.512 s	33.404 s	35.994 s
Average Time	13.52 s	21.392 s	37.191 s	38.215 s

Based on the results obtained in table 3 above, after comparing the average measured depths between the different cases, it can be deduced that the average measured depth calculated decreases as the number of particles increases. On the other hand, based on the results obtained in table 4 above, the average processing time increases as the number of particles increases. This can be explained by the fact that for a Particle Swarm Optimization problem that needs to minimize the objective function, increasing the number of particles will improve the final solution due to the presence of larger number of particles in the search space and vice versa. Thus, to achieve this solution, the PSO algorithm will require a larger processing time for a larger number of particles.

However, based on table 3, it can also be seen that the difference between the average measured depths obtained when having 15 and 25 particles is equal to 2 meters. This shows that changing the particle number from 15 to 25 has a small impact on the measured depth. However, when comparing the difference in the average measured depths calculated between 15 and 35 particles, the difference is equal to 5.86 meters. Therefore, in this case, changing the number of particles by 20 has a significant impact on the measured depth. On the other hand, based on table 4, when comparing the results for the same cases above, it can be seen that the average processing time increases significantly from 13.52 seconds to 21.392 seconds and 37.191 seconds in the cases of 15, 25, and 35 number of particles respectively.

Therefore, the selection of the number of particles largely depends on the testing scenario. If the goal is to provide the best solution regardless of the processing time, then increasing the number of particles will be the solution. However, for the purpose of this study, a balance between the solution and processing time must exist. Therefore, selecting 15 particles for the optimization problem is considered the best solution since it provides a significant decrease in the average processing time while maintaining a good average measured depth solution.

After specifying that the number of particles must be set at 15, for the same wells tested in tables 3 and 4, the impact of the convergence criteria on the measured depth and processing time was also tested. As described in section 4.2.7.4, the convergence criteria is simply the criteria that specifies when the algorithm converges. Thus, the value of the difference in the objective function between two particles of the same iteration will be calculated. For the PSO optimizer being implemented in this study, the convergence criteria is calculated based on the average particle and best particle of the same iteration. Once this difference reaches the convergence criteria, the algorithm will terminate and provide the optimal solution. The results obtained are found in the tables below:

Table 5 Measured depth results after changing the convergence criteria.

Wells	Convergence Criteria = 0.01 Measured Depth	Convergence Criteria = 0.05 Measured Depth	Convergence Criteria = 0.1 Measured Depth	Convergence Criteria = 0.3 Measured Depth
1	2555.844 m	2558.368 m	2555.142 m	2559.218 m
2	2547.401 m	2553.863 m	2551.106 m	2549.235 m
3	2528.480 m	2531.859 m	2545.638 m	2531.988 m
4	2464.580 m	2468.384 m	2465.887 m	2465.501 m
5	2707.768 m	2708.079 m	2708.458 m	2755.629 m
6	2646.0795 m	2636.989 m	2694.934 m	2702.312 m
7	Not Feasible	Not Feasible	Not Feasible	Not Feasible
8	2917.597 m	2909.583 m	2899.946 m	2941.145 m
9	2482.192 m	2483.285 m	2482.922 m	2493.040 m
10	2628.687 m	2629.030 m	2629.092 m	2635.039 m
Average	2608.737 m	2608.827 m	2614.792 m	2625.901 m

Table 6 Processing times results after changing the convergence criteria.

Well Number	Convergence Criteria: 0.01 Processing time	Convergence Criteria: 0.05 Processing time	Convergence Criteria: 0.01 Processing time	Convergence Criteria: 0.03 Processing time
1	18.522 s	11.338 s	7.468 s	5.105 s
2	12.951 s	10.480 s	8.401 s	5.756 s
3	16.174 s	8.952 s	2.983 s	4.406 s
4	18.040 s	9.794 s	7.237 s	5.602 s

5	17.588 s	8.629 s	5.284 s	2.111 s
6	0.696 s	0.548 s	0.569 s	0.826 s
7	Not Feasible	Not feasible	Not Feasible	Not Feasible
8	5.949 s	3.923 s	5.527 s	2.531 s
9	17.548 s	7.831 s	8.067 s	4.636 s
10	14.214 s	8.081 s	6.472 s	4.773 s
Average Time	13.52 s	7.73 s	5.778 s	3.971 s

Based on the results of table 5, the average measured depth obtained increases with the increase in the convergence criteria. On the other hand, after comparing the results of table 6, it can be deduced that the average processing time decreases with the increase in the converging criteria. This can be explained by the fact that, as stated previously, the convergence criteria is simply the difference between the objective function of particles in the same iteration. Thus, as the convergence criteria is increased, the solution of the optimization algorithm does not improve and vice versa. However, as the convergence criteria increases, less processing time is needed for the algorithm to converge.

According to table 5, the difference in the average measured depth for the cases having the convergence criteria of 0.01 and 0.05 is less than 1 meter. This indicates that, in this specific case, increasing the convergence criteria by 0.04 has little impact on the calculated measured depth. However, the difference in the average measured depth between the cases having the convergence criteria of 0.01 and 0.3 is equal to 6.05 meters. Thus, increasing the convergence criteria by 0.29 will lead to a significant change in the calculated objective function. On the other hand, based on table 6, when comparing the results for the same cases above, it can be seen that the average processing time decreases significantly from 13.52 seconds to 7.73 seconds and 3.971 seconds for the cases having the convergence criteria of 0.01, 0.05, and 0.3 respectively.

As a summary, the selection of the convergence criteria largely depends on the testing scenario. If the goal is to provide the best solution regardless of the processing time, then decreasing the convergence criteria will be the solution. However, for the purpose of this study, a balance between the solution and processing time must exist. Therefore, selecting a convergence criteria of 0.05 for the optimization problem is considered the best solution since it provides an almost exact average measured depth when compared to the lowest convergence criteria of 0.01 while simultaneously decreasing the processing time by approximately 43%.

Finally, after defining that the swarm size must be set at 15 and the convergence criteria must not exceed 0.05, the maximum number of iterations must be specified. Thus, after fixing the swarm size and convergence criteria at 15 and 0.05 respectively, the first five wells presented in the sensitivity analysis above were tested based on a maximum iteration of 100. The results will be presented in the figures below.

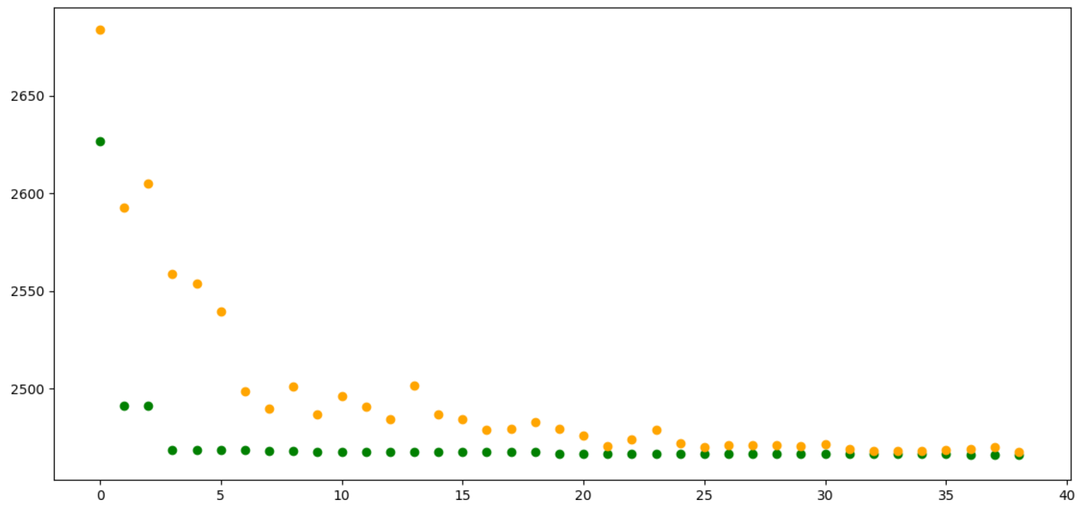


Figure 19 Well 1.

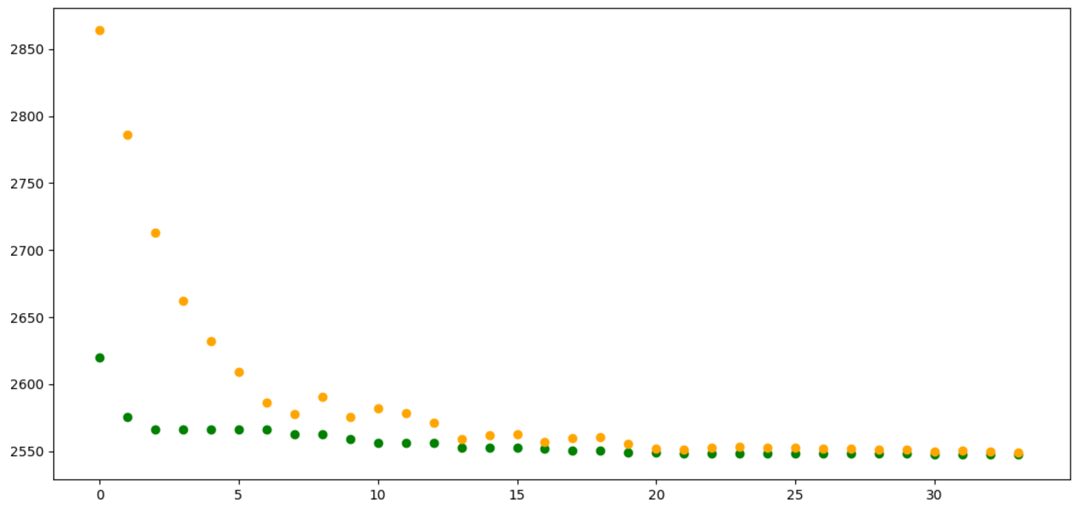


Figure 21 Well 2.

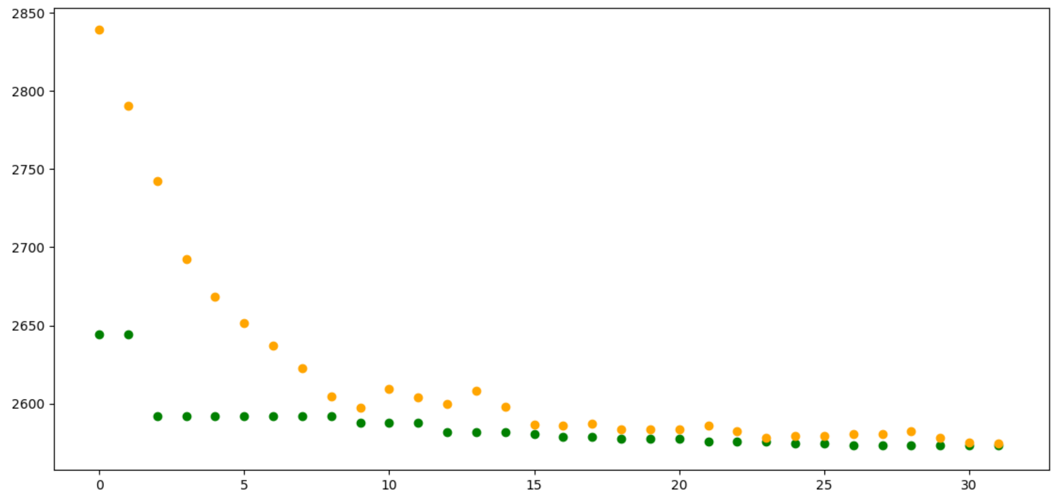


Figure 20 Well 3.

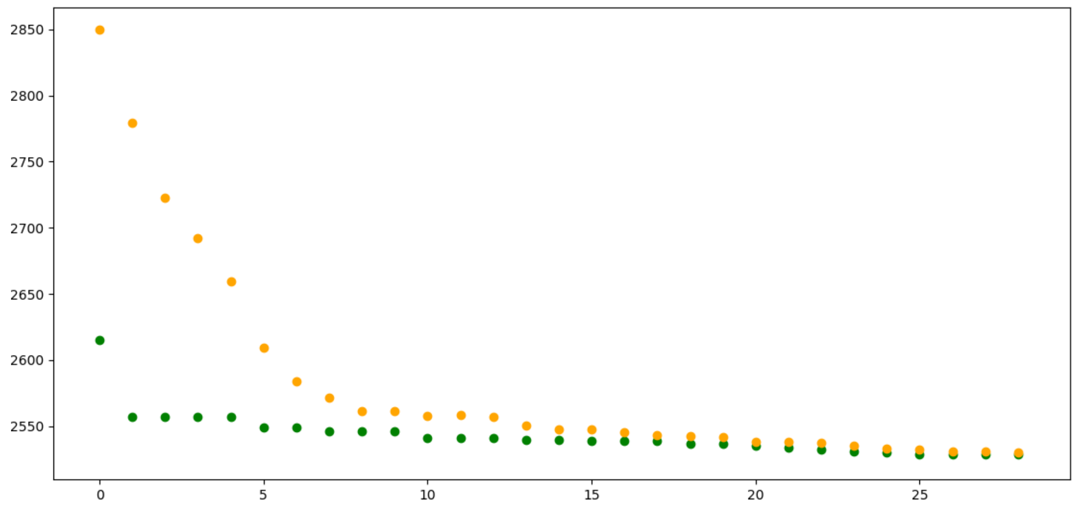


Figure 22 Well 4.

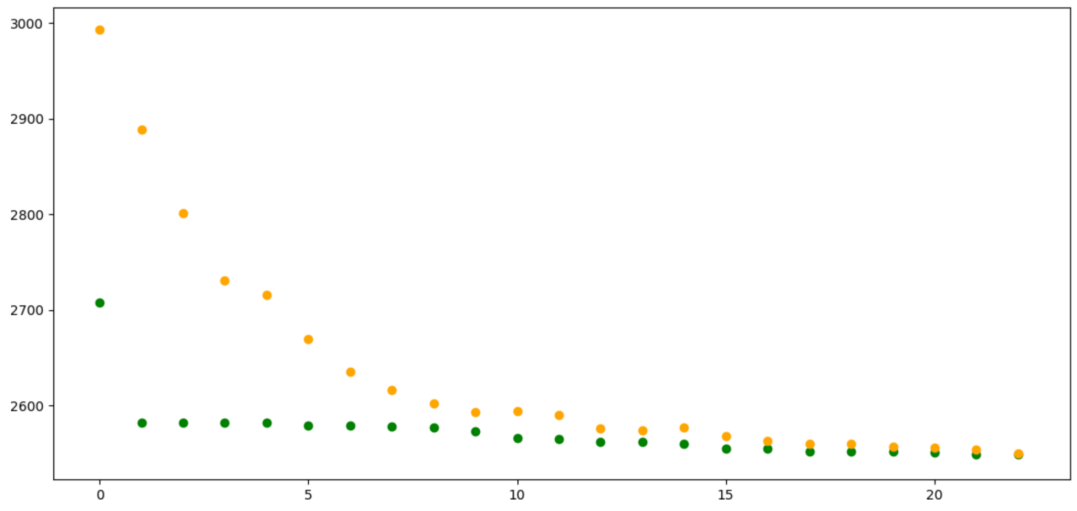


Figure 23 Well 5.

In the figures above, the x-axis and y-axis simply represent the number of iterations and objective function values respectively. In addition, the green and orange particles represent the average particle and best particle for each iteration in the PSO algorithm. Thus, for an iteration of 0, two particles are seen. The green particle symbolizes the particle with the lowest objective function value. While as, the orange particle, represents the average objective function value obtained between the particles. Once the average particle approaches the green particle based on the convergence criteria, the algorithm converges.

According to figures 19, 20, 21, 22 and 23 it is evident that the algorithm is converging at a maximum of 36 iterations. Thus, it was decided to implement 100 as the maximum number of iterations to provide the algorithm with some added space in case of no convergence. Any number above this value will be redundant since most wells are converging when the number of iterations is reaching 40.

6.2. Testing Results and Discussion.

In this section, 24 different cases were tested by implementing the optimization parameters obtained in section 6.1. The graphical results of these cases will be presented in this section below. It is important to note that, the visualization of three different results will take place in this section. These results are divided into three sections.

6.2.1. Initial trajectory

In this section, the trajectories produced are based on the Euclidean distance from the drilling center to reach the point of intersection.

6.2.2. Manual optimization

In this case, the trajectories produced are based on fixing the kick-off point and the arbitrary parameter to reach Cs at 400 meters and 600 meters respectively. For the parametric t, a while loop is introduced that loops from [0,200]. This loop will terminate once a feasible solution is obtained.

6.2.3. Automated optimization.

The results obtained from this case are simply the trajectories produced by implementing the different optimization parameters defined in section 6.1.

6.2.4. Results and discussion.

The results of 24 cases will be presented in this thesis paper. These results will combine the three sections (6.2.1, 6.2.2, and 6.2.3) in the same figure for cases 1, 2, 4, and 5. Otherwise, all other cases will only contain figures combining sections 6.2.1 and 6.2.3. Thus, every figure will represent a case. However, specifically for case 1, four figures will be presented. The first three figures will show the three different sections separately. While the fourth case will combine all three sections together.

6.2.4.1. Case 1.

Case 1 includes 20 targeted vertical wells. The results obtained in this case are shown in the figures below.

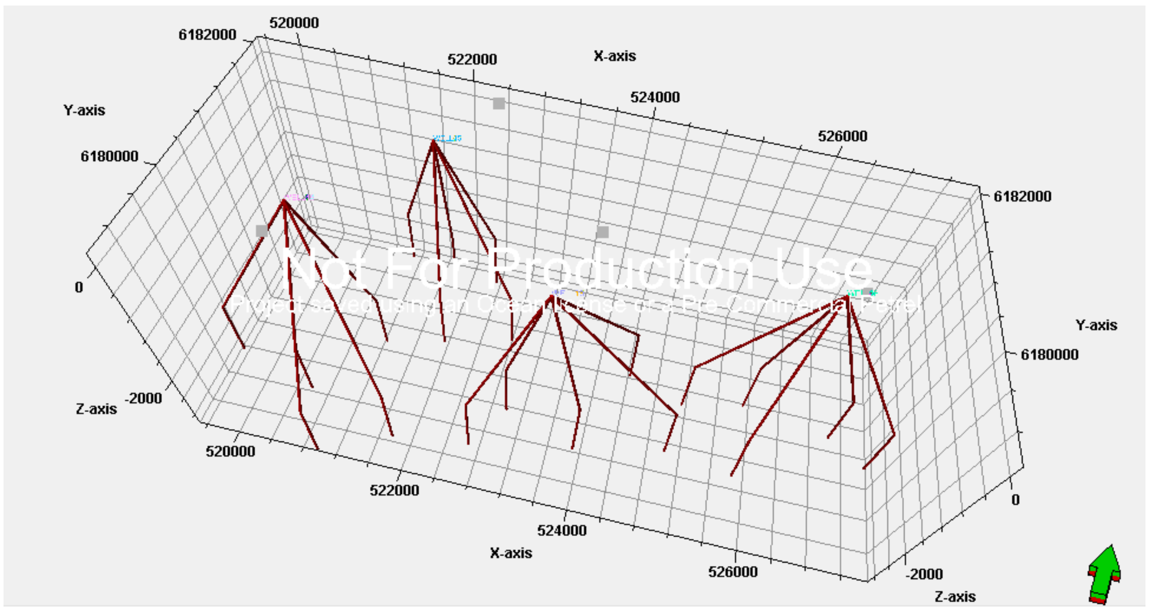


Figure 24 Vertical trajectories of case 1.

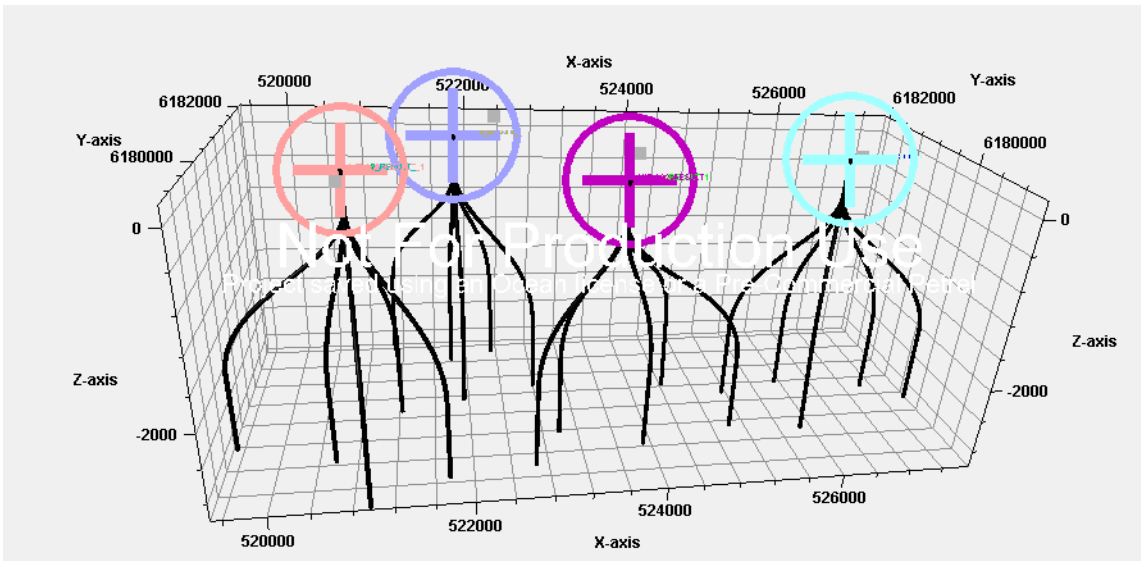


Figure 25 Manual optimized trajectories of case 1.

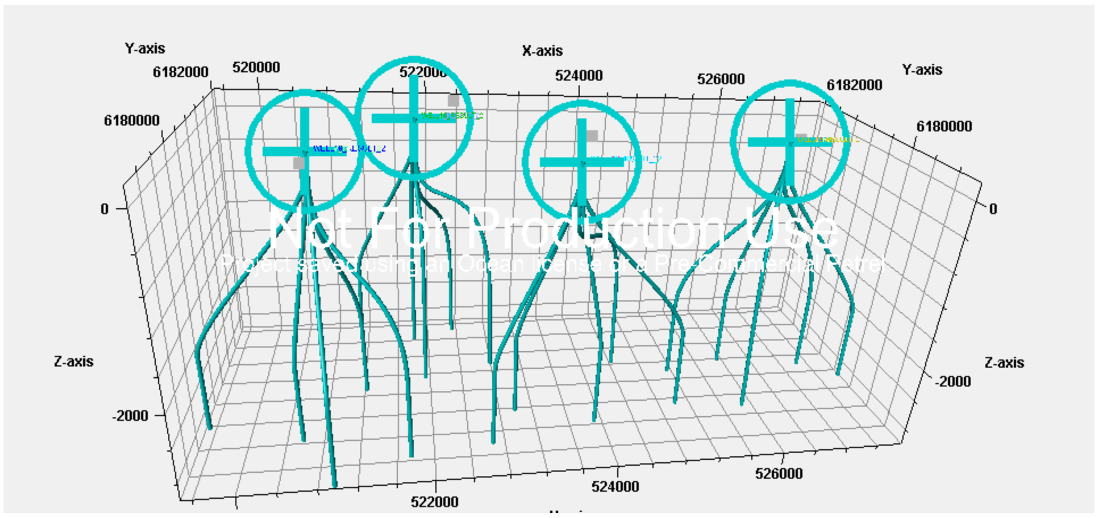


Figure 26 Automated optimized trajectories of case 1.

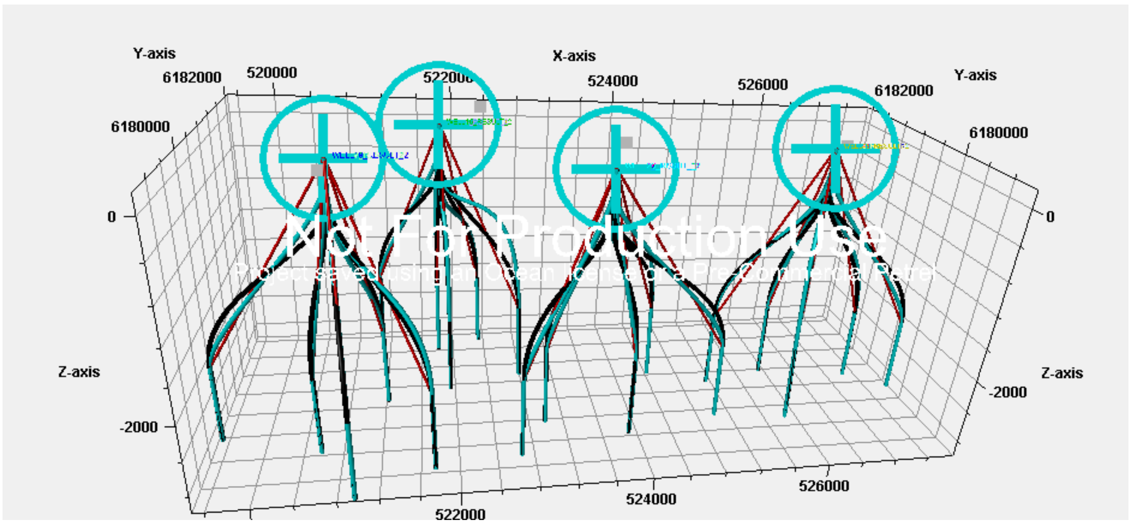


Figure 27 Case 1.

In figure 24, the initial trajectory is represented by red. However, in figure 25, the manual trajectory is represented by black. Lastly, based on figure 26, the optimized trajectory is designated by blue. For consistency, all results will follow the same color scheme as in this case.

According to figure 24, the initial trajectory is simply the straight distance that connects the drilling center with the intersection point. This is the initial trajectory that is provided to the code as an input (found in section 4.7.2.2.1). It is obvious that this type of trajectory may not be used in real drilling cases due to the high degree of bending. To solve this issue, the usage of the Bezier curve algorithm was implemented.

Based on figures 25, 26 and 27, it can be deduced that curved smooth trajectories are produced when applying the designed algorithm. However, to be able to assess the difference between the manual trajectories and optimized trajectories, the following results must be compared in the table below:

Table 7 Measured depth comparison for manual and optimized trajectories.

Well Number:	Manual Trjaectory Measured Depth	Optimized Trajectory Measured Depth
1	2407.029 m	2367.532 m
2	2287.922 m	2331.316 m
3	2616.061 m	2543.55 m
4	2563.624 m	2560.813 m
5	2582.243 m	2561.305 m
6	2467.466 m	2529.286 m
7	2280.918 m	2265.161 m
8	2699.738 m	2611.781 m
9	2336.723 m	2366.378 m
10	2622.866 m	2652.013 m
Average MD	2486.459 m	2478.913 m

As seen in table 7 above, the implementation of PSO optimization on the well trajectory algorithm decreased the average measured depth calculated by 7.54 meters when compared to the manual trajectory. This indicates that the implementation of the

PSO algorithm has proven to be beneficial and will aid in reducing the cost associated in well designing and drilling.

6.2.4.2. Case 2.

Similar to case 1, the wells that need to be intersected are vertical wells in this case. The resulting trajectories obtained are found in the figure below:

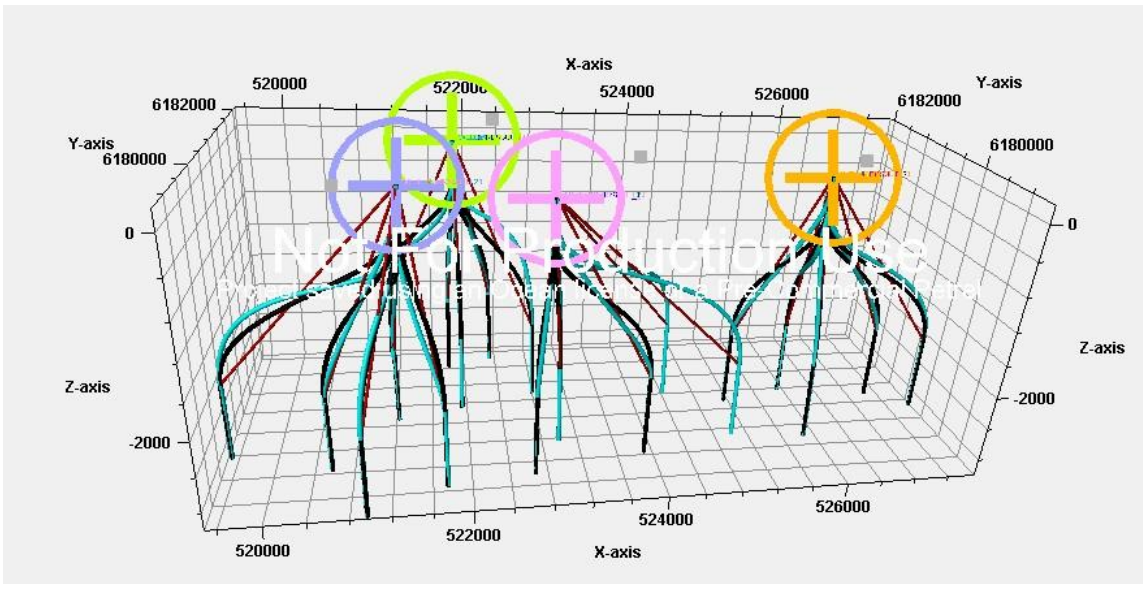


Figure 28 Case 2.

6.2.4.3. Case 3.

The wells that need to be intersected in this case are vertical wells. The results obtained are visualized in the figure below:

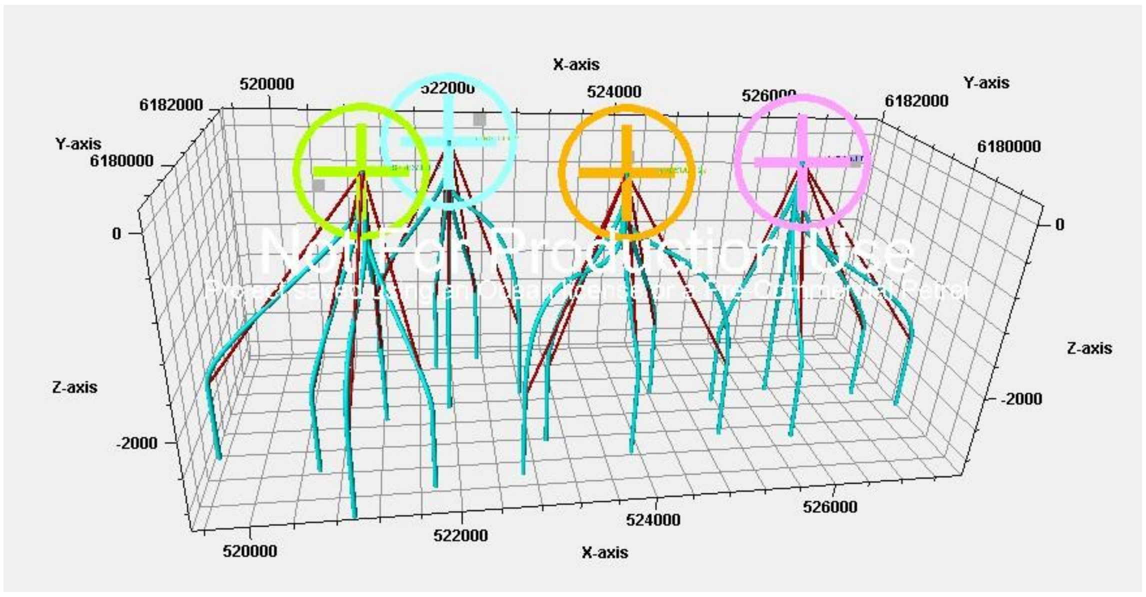


Figure 29 Case 3.

6.2.4.4. Case 4.

In this case, the targeted wells are horizontal wells. The results obtained will be shown in the figure below:

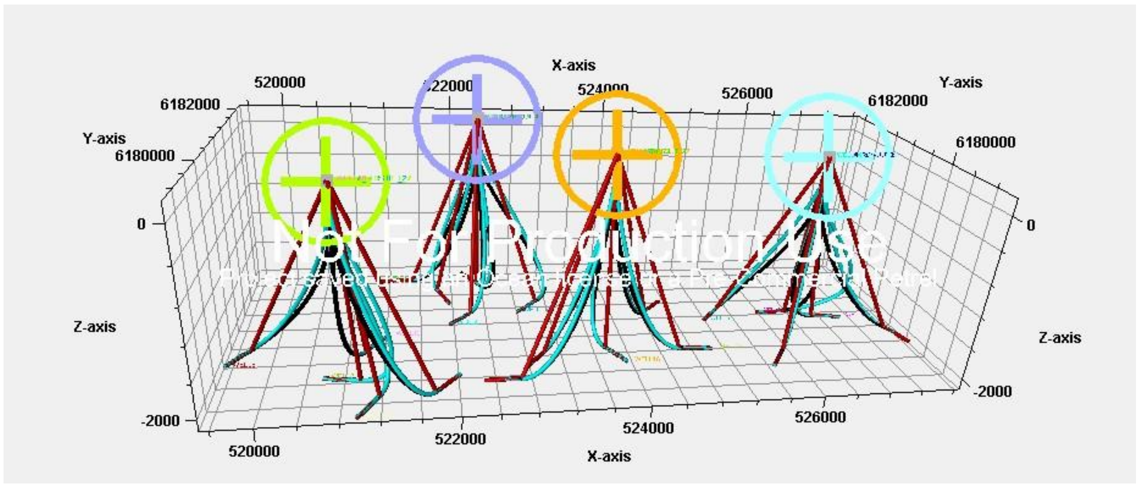


Figure 30 Case 4.

To be able to assess the difference between the manual trajectories and optimized trajectories in the case of horizontal targets, the following results must be compared in the table below:

Table 8 Measured depth comparison for manual and optimized trajectories in the case of horizontal targets.

Well Number:	Manual Trjaectory Measured Depth	Optimized Trajectory Measured Depth
1	2572.052 m	2558.954 m
2	2565.069 m	2537.929 m
3	2545.312 m	2520.335 m
4	2468.793 m	2463.954 m
5	2724.818 m	2685.138 m
6	2595.39 m	2508.007 m
7	Not Feasible	2544.65 m
8	2924.206 m	2861.345 m
9	2493.627 m	2480.567 m
10	2646.694 m	2612.647 m
Average MD	2615.107 m	2580.986 m (excluding well 7).

Based on the results obtained in table 8, it can be deduced that the usage of the PSO optimizer was able to decrease the average measured depth by 34.12 meters. This shows the significance of implementing the optimizer to minimize the measured depth needed to intersect the target. In addition, as seen in table 8, well number 7 was initially unfeasible in the case of manual trajectory. However, after the introduction of the PSO optimizer, the well became feasible. This further shows the associated improvement in implementing the optimization algorithm.

6.2.4.5. Case 5.

The target wells of this case are horizontal wells. The results are shown in the figure below:

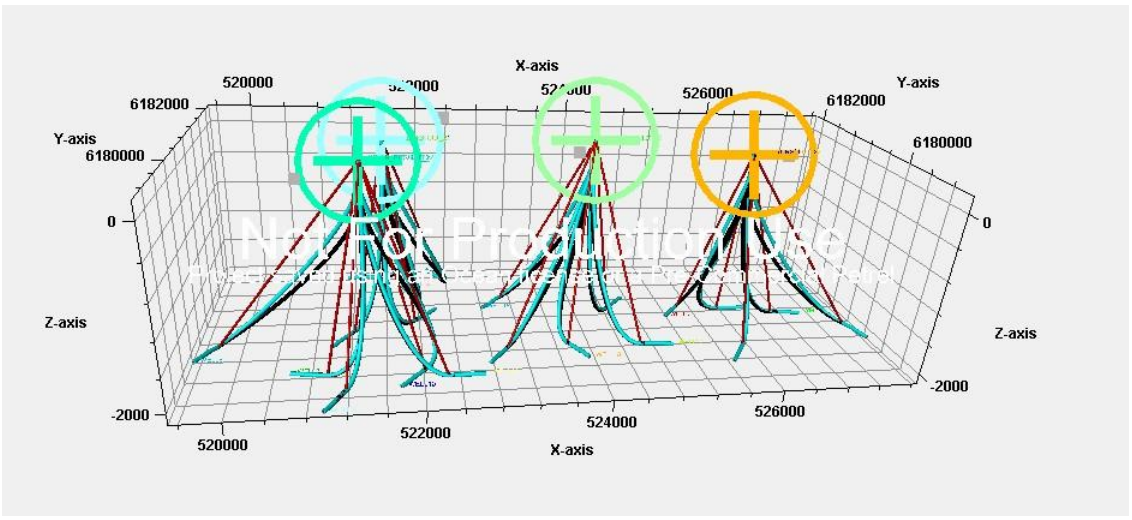


Figure 31 Case 5.

6.2.4.6. Case 6.

The results obtained in case 6 are shown in the figure below:

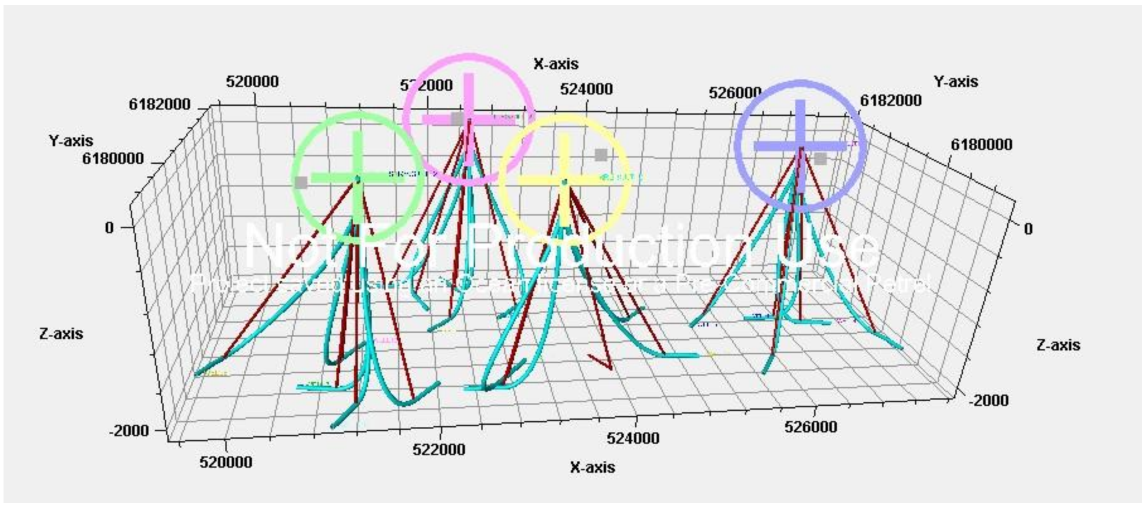


Figure 32 Case 6.

6.2.4.7. Case 7.

The obtained trajectories in this case are shown in the figure below:

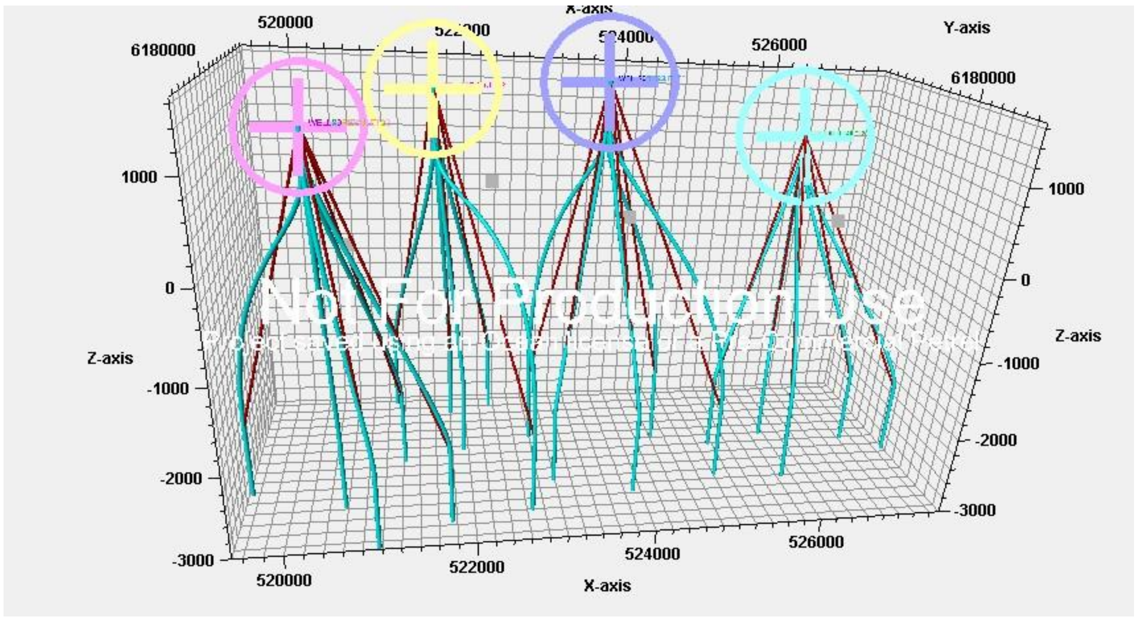


Figure 33 Case 7.

6.2.4.8. Case 8.

The calculated trajectories are found in the figure below:

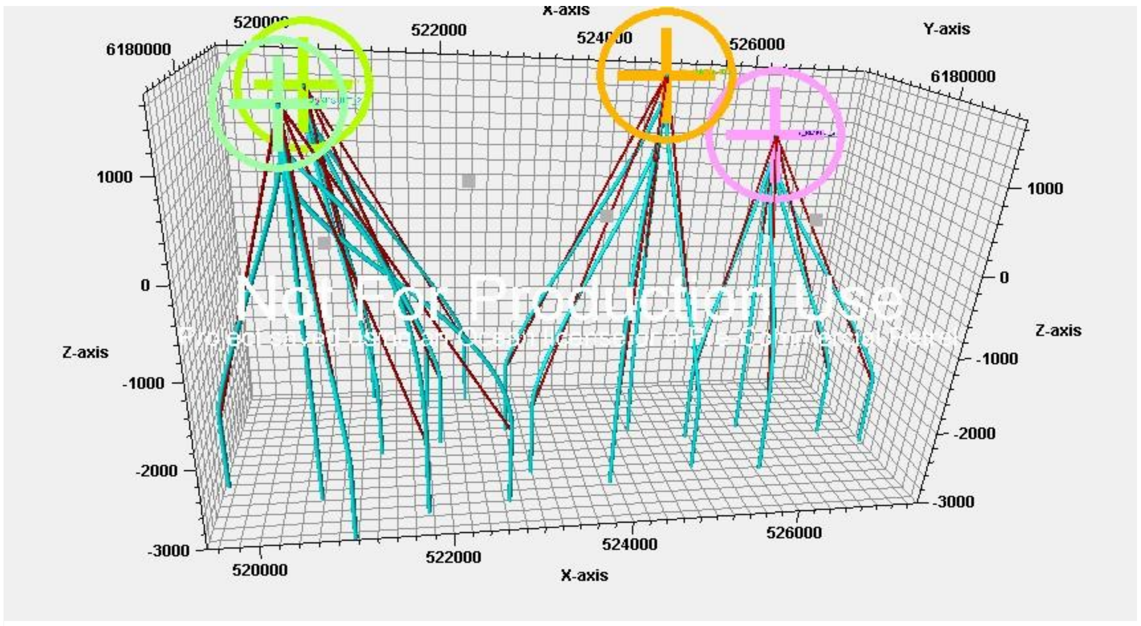


Figure 34 Case 8.

6.2.4.9. Case 9.

The calculated trajectories are found in the figure below:

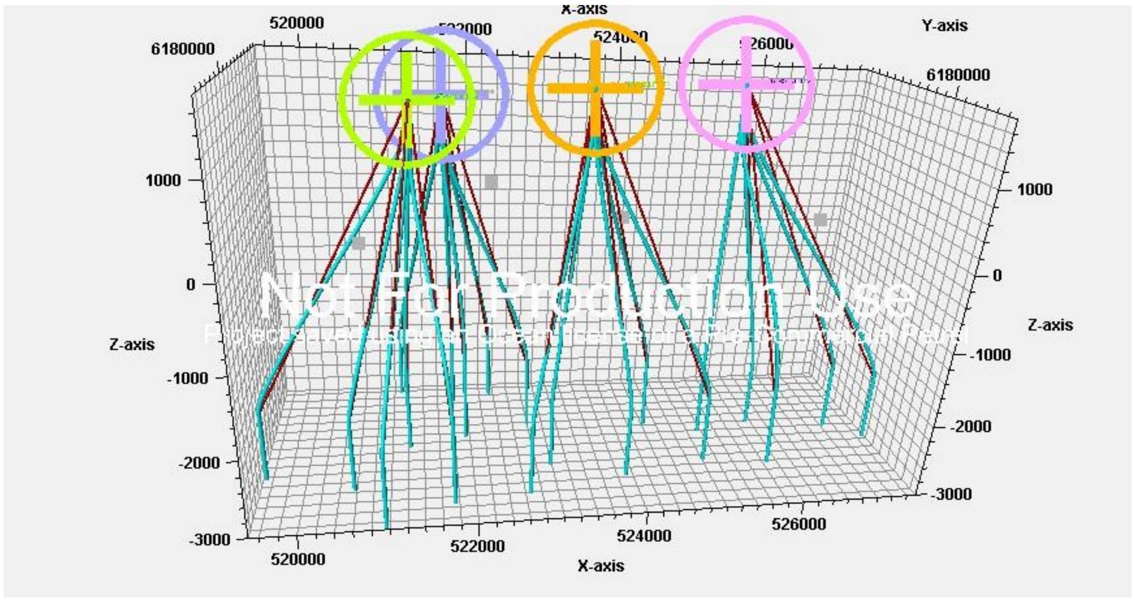


Figure 35 Case 9.

6.2.4.10. Case 10.

The calculated trajectories are found in the figure below:

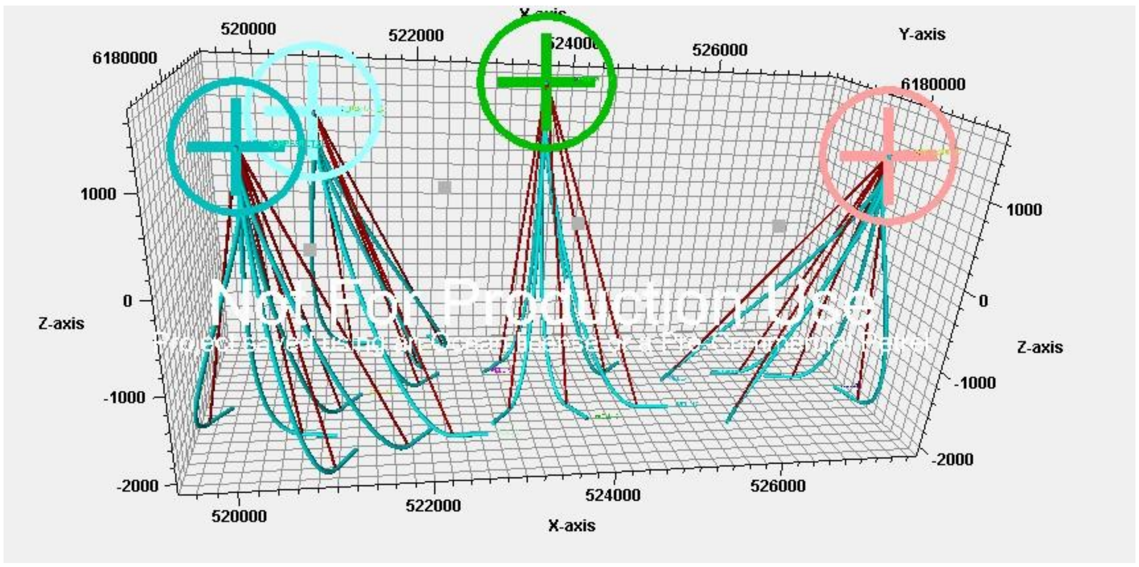


Figure 36 Case 10.

6.2.4.11. Case 11.

The calculated trajectories are found in the figure below:

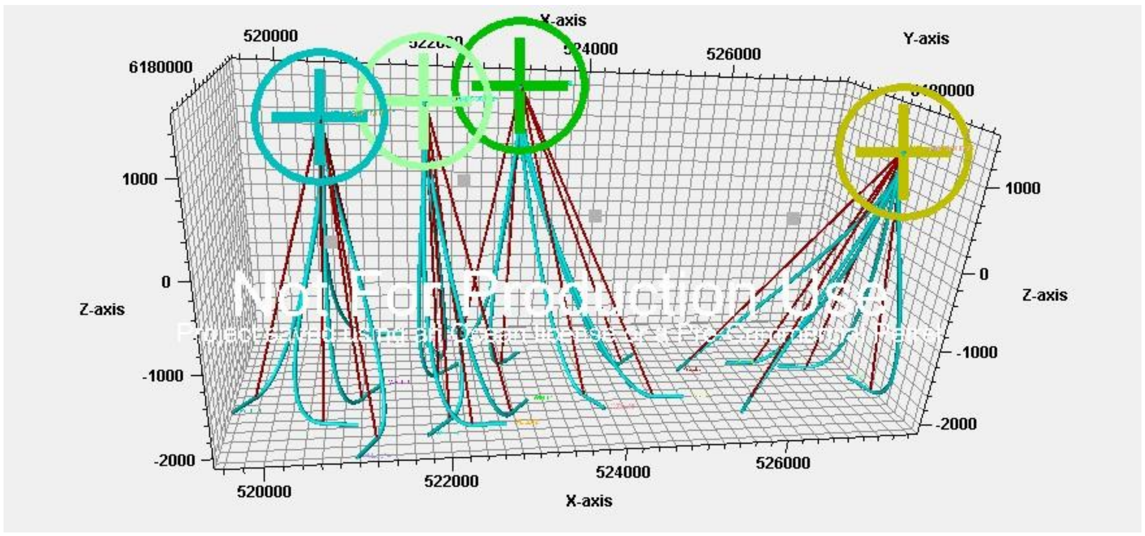


Figure 37 Case 11.

<u>6.2.4.12.</u> <u>Case 12.</u>

The calculated trajectories are found in the figure below:

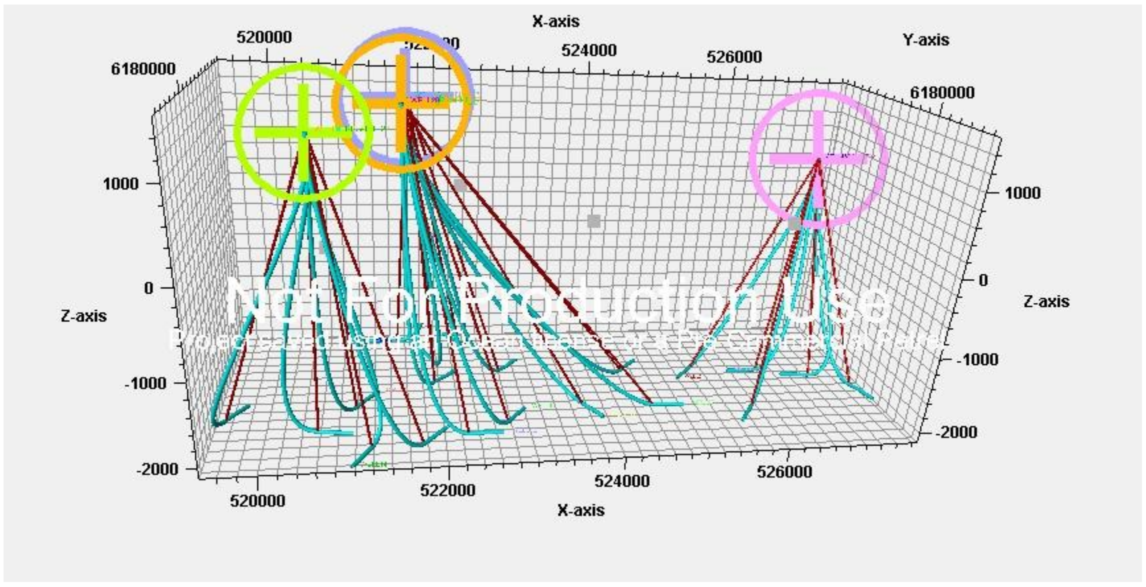


Figure 38 Case 12.

<u>6.2.4.13.</u> <u>Case 13.</u>

The calculated trajectories are found in the figure below:

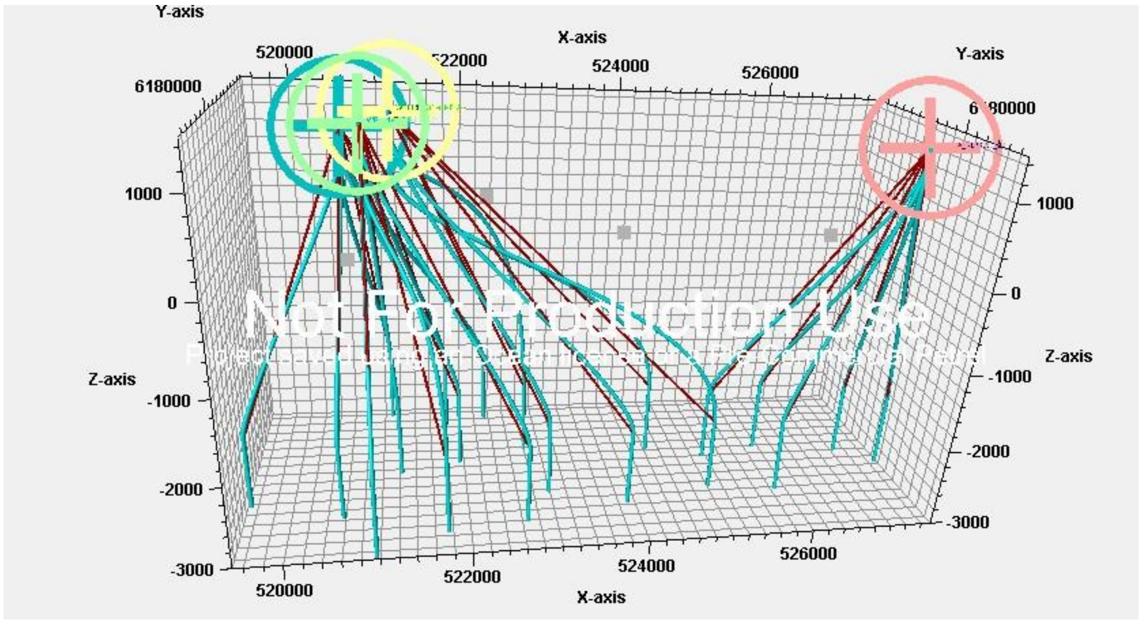


Figure 39 Case 13.

6.2.4.14. Case 14.

The calculated trajectories are found in the figure below:

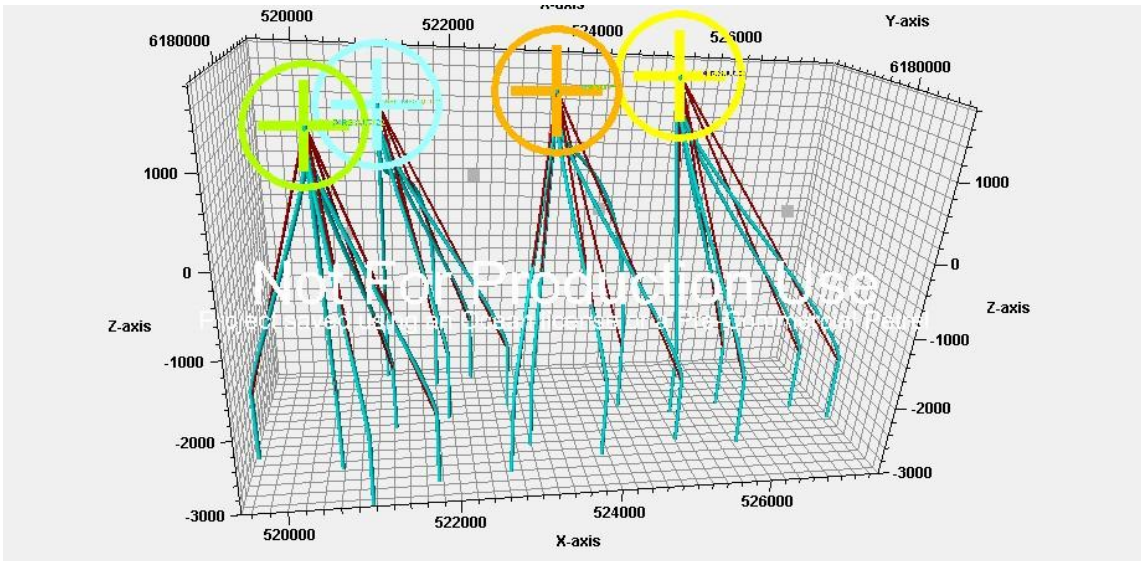


Figure 40 Case 14.

6.2.4.15. Case 15.

The calculated trajectories are found in the figure below:

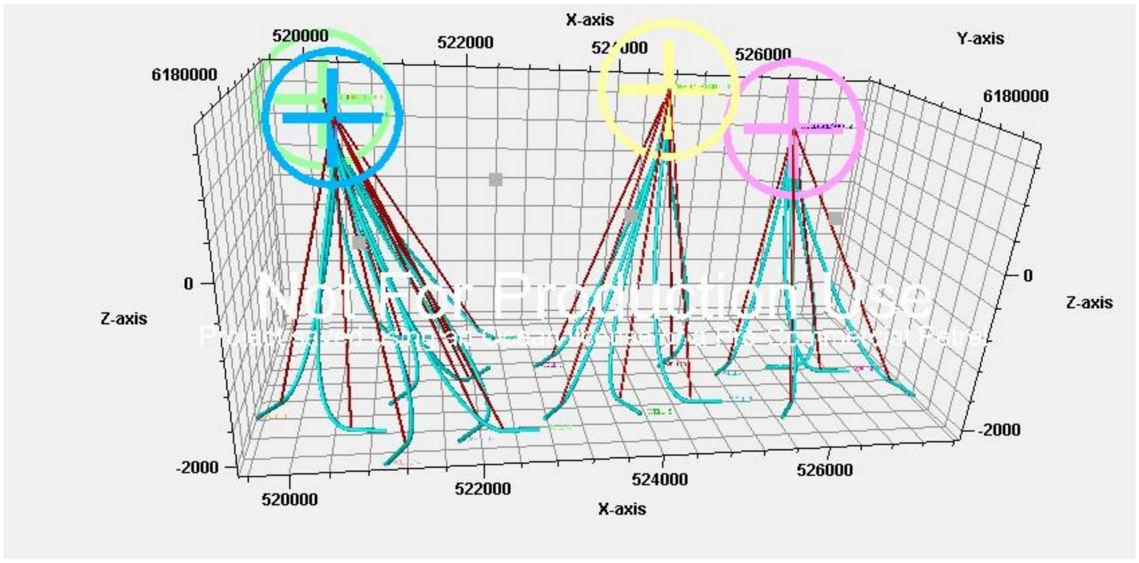


Figure 41 Case 15.

6.2.4.16. Case 16.

The calculated trajectories are found in the figure below:

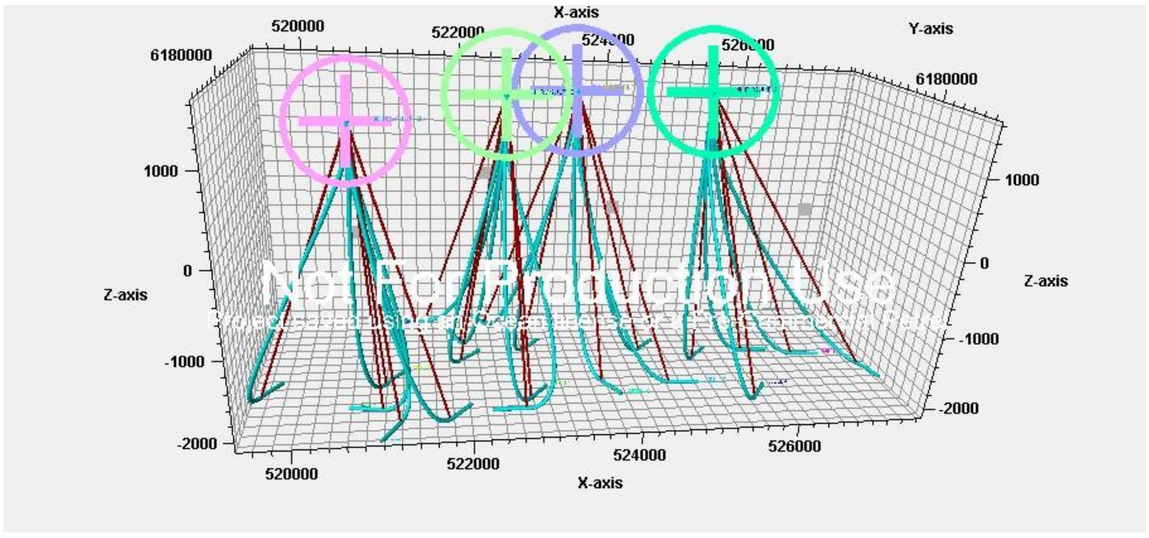


Figure 42 Case 16.

6.2.4.17. Case 17.

The calculated trajectories are found in the figure below:

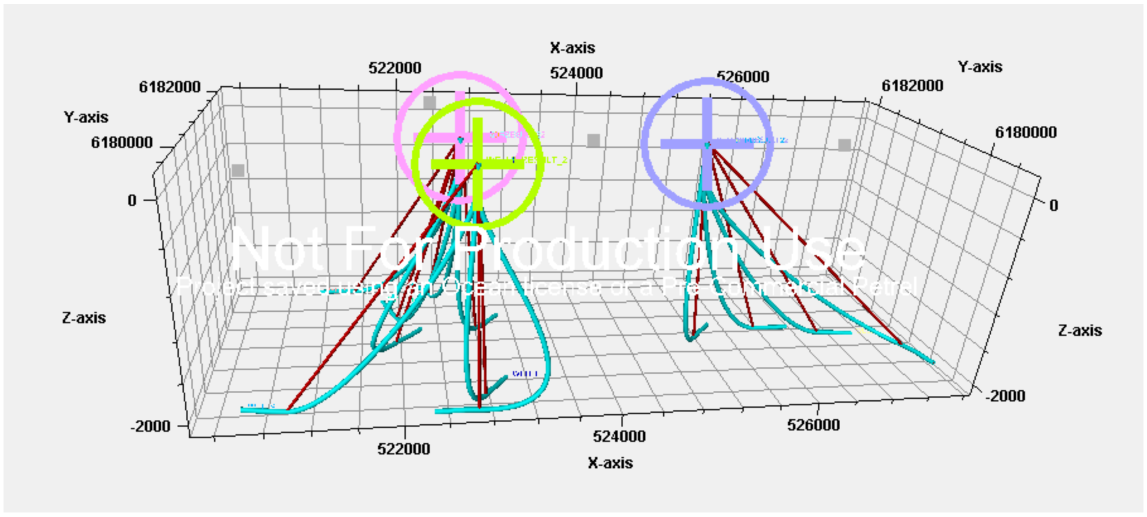


Figure 43 Case 17.

6.2.4.18. Case 18.

The calculated trajectories are found in the figure below:

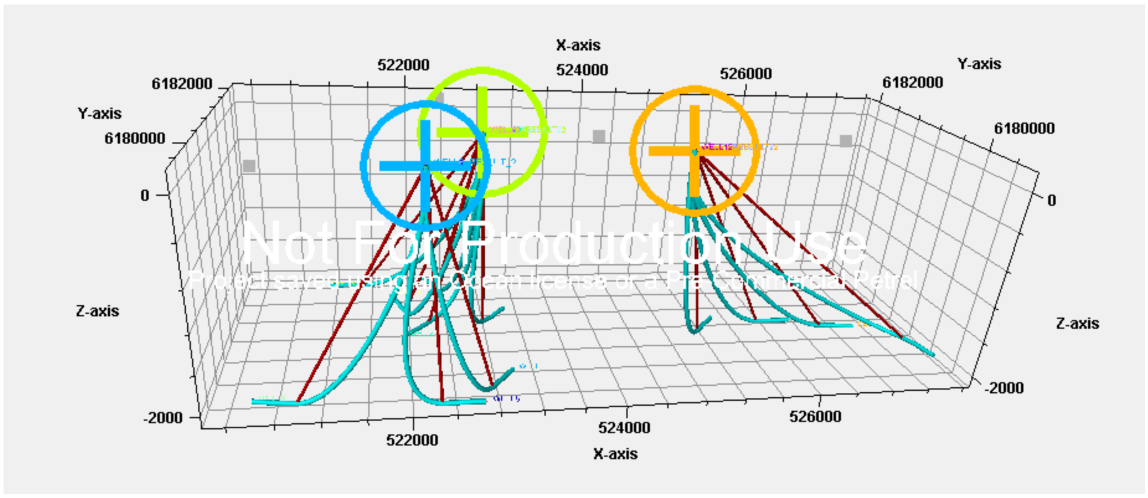


Figure 44 Case 18.

6.2.4.19. Case 19.

The calculated trajectories are found in the figure below:

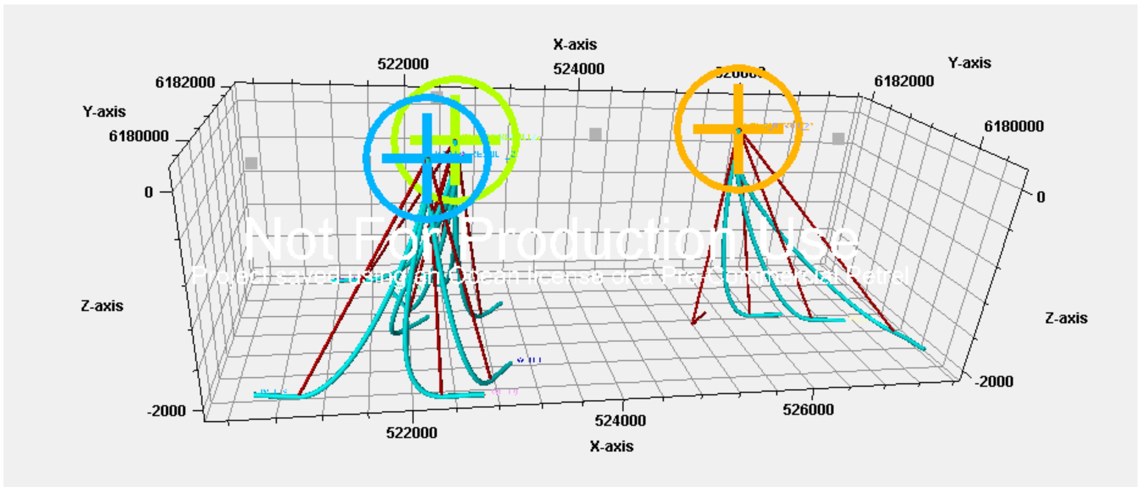


Figure 45 Case 19.

6.2.4.20. Case 20.

The calculated trajectories are found in the figure below:

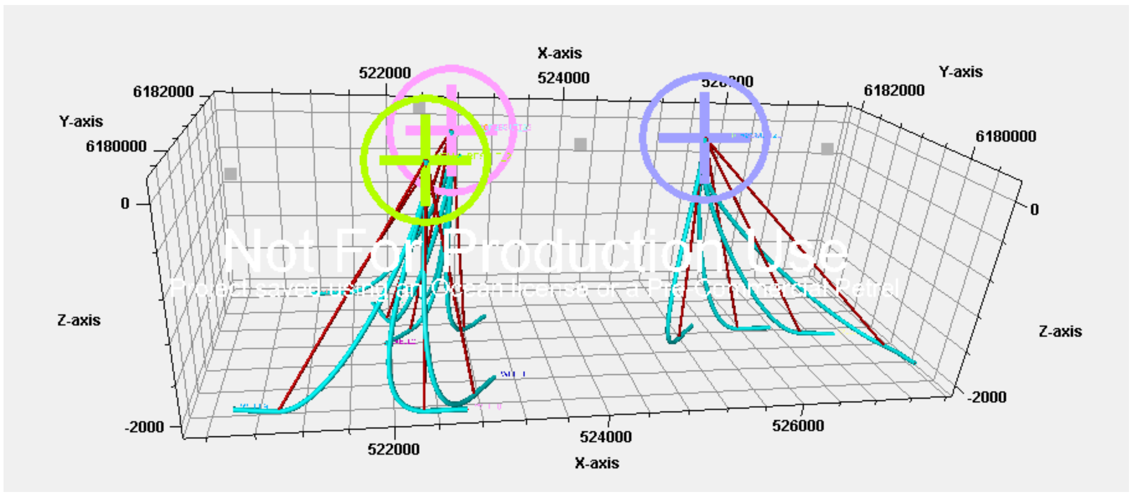


Figure 46 Case 20.

6.2.4.21. Case 21.

The calculated trajectories are found in the figure below:

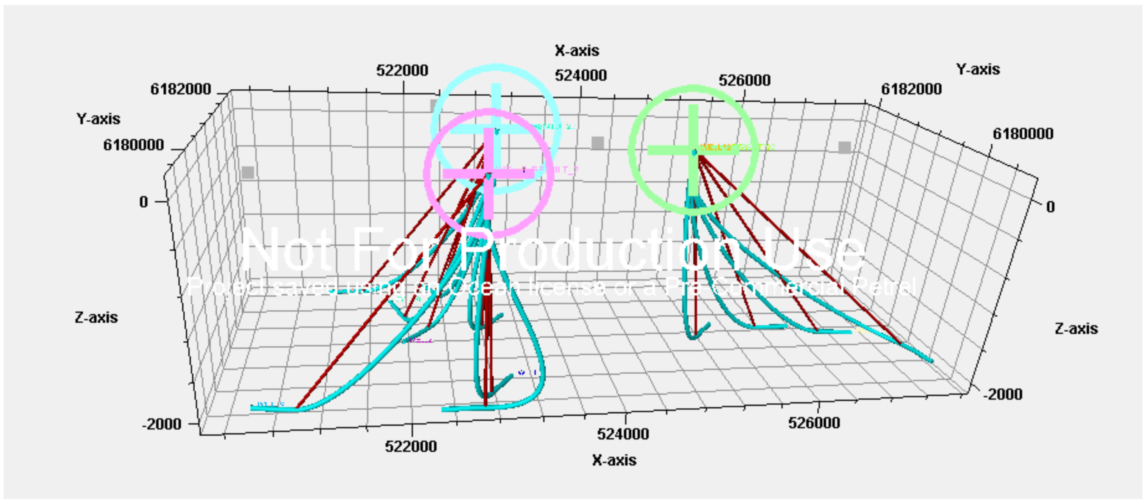


Figure 47 Case 21.

<u>6.2.4.22.</u> <u>Case 22.</u>

The calculated trajectories are found in the figure below:

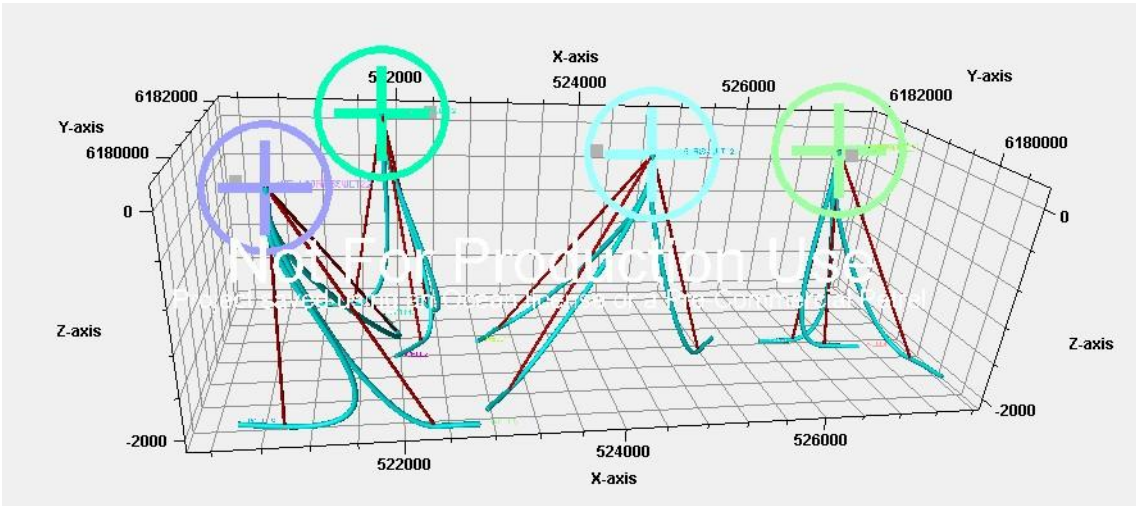


Figure 48 Case 22.

<u>6.2.4.23.</u> <u>Case 23.</u>

The calculated trajectories are found in the figure below:

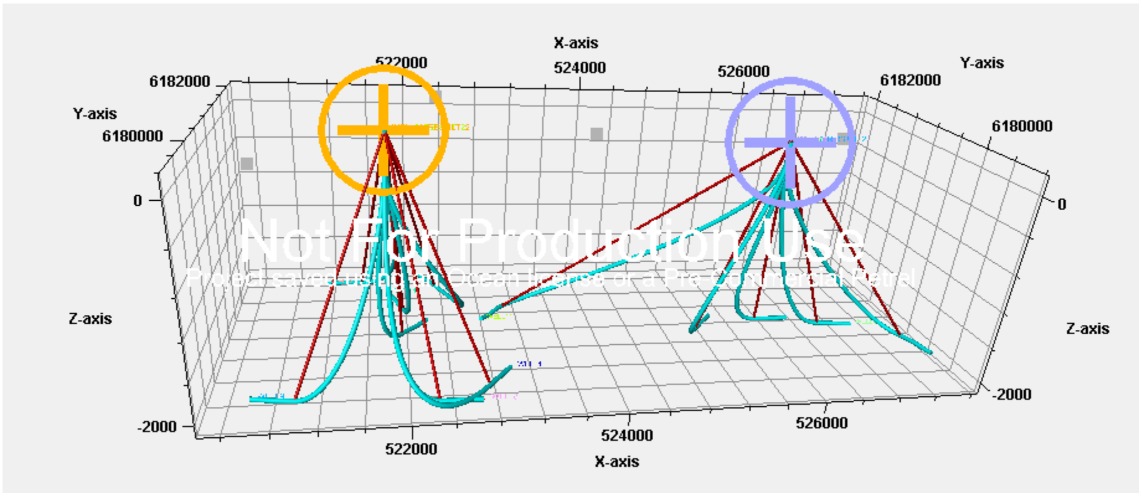


Figure 49 Case 23.

6.2.4.24. Case 24.

The calculated trajectories are found in the figure below:

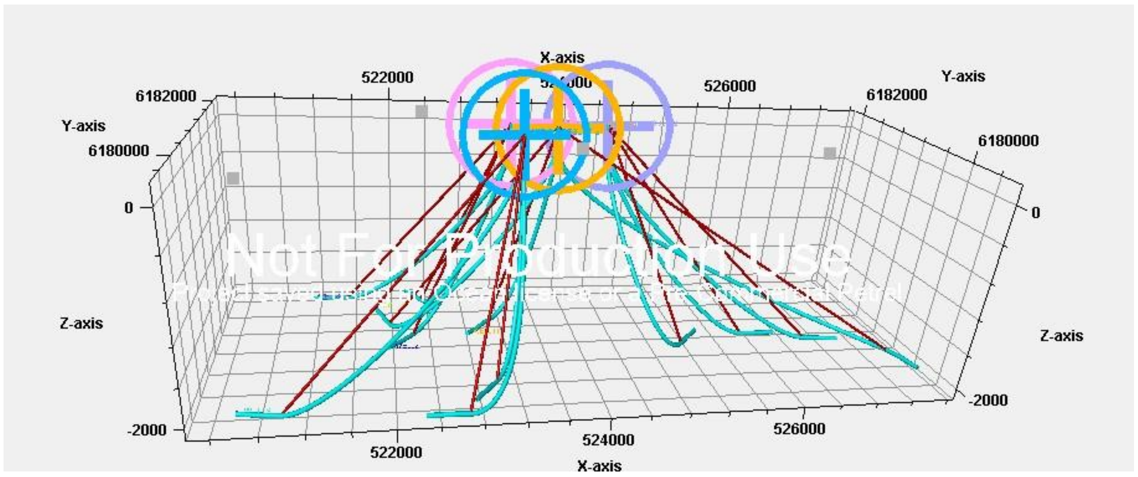


Figure 50 Case 24.

As a summary, based on the results obtained in section 6.2.4, it can be deduced that the usage of the Bezier curve function to model well trajectories was able to smoothen the well path. This usage allowed the conversion of unfeasible wells to feasible wells by decreasing the associated dogleg severity. In addition, to further improve the Bezier curve function, the particle swarm optimization algorithm was introduced. The optimized trajectories obtained were compared to the trajectories obtained from the manual scenario. Based on this comparison, it was shown that the average measured depth of the optimized trajectory was decreased when compared to the manual trajectory. This decrease was more significant in the case of horizontal well targets.

CHAPTER 7

VERTIFICATION WITH PETREL

To further support the importance of the Bezier curve algorithm, it was compared to the well path algorithm implemented in Petrel. The results obtained will be presented in this chapter.

7.1. Well Path Algorithm in Petrel.

This algorithm implements the Advanced Design Trajectory (ADT) to generate well trajectories (SCM,2009). To design trajectories and calculate costs, the algorithm will limit the curvature to a specified DLS (SCM,2009). The user must provide the algorithm with the upper and lower DLS limit. Also, both the starting point and reservoir target must be provided (SCM, 2009). Finally, the user has the option to provide the exact measured depth to reach the kick-off point.

7.2. Input Parameters Set in Petrel.

For this comparison to be efficient, the input provided to Petrel consisted of the following:

1- Same starting point or well head as in the case of the Bezier curve testing case.

2- Same reservoir target as in the case of the Bezier curve testing case.

3- Same measured depth to reach kick-off point as the Bezier curve testing case.

4- Requested upper DLS was set at 7.

5- Lower DLS limit was set at 3. (Since this algorithm will calculate trajectories by using either the low DLS limit or the upper DLS limit, a trade-off must be established between having a very low DLS limit which will yield very large measured depths to intersect the target and having a very high DLS limit which will yield excessive tension on the drilling equipment).

7.3. Results of Testing Wells.

10 different wells presented in case 4 of chapter 6 were tested using both the Bezier curve algorithm and Petrel Well Path algorithm. The results are shown in the figure below:

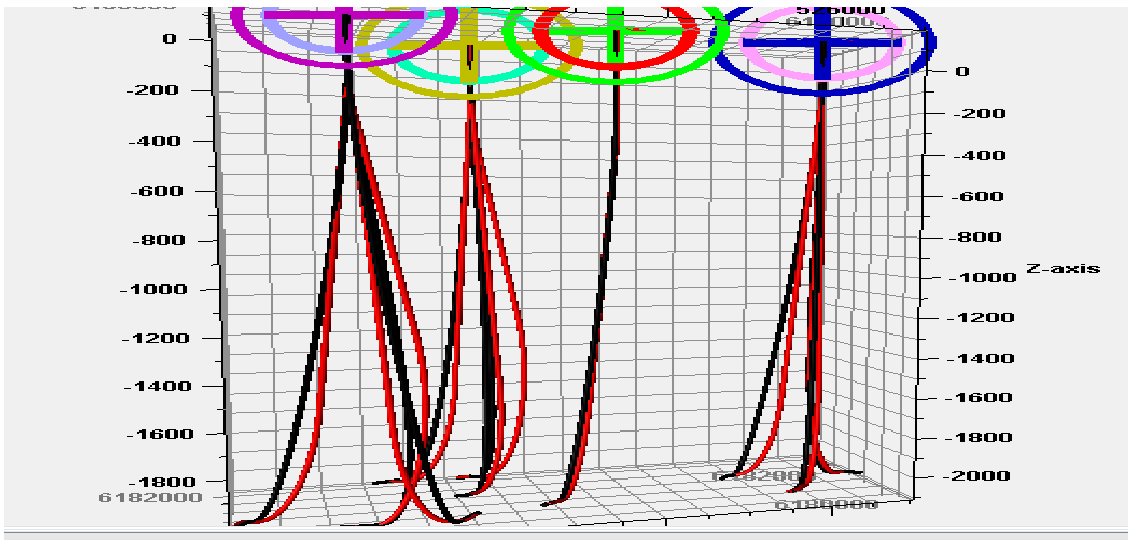


Figure 51 Results.

The black and red curves presented in figure 51 represent the results using the Bezier curve and Petrel Well Path algorithms respectively. Based on this figure, it is evident that by using the Bezier curve algorithm, shorter curves were obtained when compared to Petrel's Well Path Algorithm.

To further support this claim, the average measured depth and average CPU needed to produce trajectories using both models will be presented in the two tables below.

Table 9 Comparison between Bezier algorithm and Petrel.

Algorithm Used	**Total Average Measured Depth (Meters) for all wells**
Petrel's algorithm	2989.224 m
Bezier algorithm	2977.82 m

Table 10 Comparison between Bezier algorithm and Petrel.

Algorithm Used	**Average Maximum DLS reached**
Petrel's algorithm	7
Bezier algorithm	6.96

Based on table 9, it can be deduced that using the Bezier curve algorithm aided in decreasing the total average measured depth by 11.4 meters. Thus, it is safe to say that this algorithm provided shorter wells when compared to Petrel.

Based on table 10, it can also be deduced that the Bezier curve algorithm needed a slightly lower average maximum DLS limit of 6.96 when compared to Petrel which obtained an average maximum DLS average of 7. This shows that the Bezier curve algorithm will generate wells that help in decreasing the tension exerted on the wellbore and the drilling equipment. Thus, using this algorithm was proven to be efficient and economical.

CHAPTER 8

CONCLUSION AND RECOMMENDATIONS FOR FUTURE WORK

This chapter will include the conclusion and a few suggestions on how to further develop this thesis in the future.

8.1. Conclusion.

The objective of this thesis has been to design and implement a well planning algorithm that is capable of producing 3D well trajectories. To achieve this purpose, an extensive literature review was conducted to search for the best model that would provide simplicity and added flexibility. Based on the research conducted in this section, the Bezier curve model was chosen for this purpose. Once the model was selected, the algorithm was designed.

After the algorithm was designed, it was deduced that the optimization of the three input parameters associated with the Bezier model would provide optimal and robust solutions. Therefore, the particle swarm optimization technique was implemented. For this optimization problem, after conducting a sensitivity analysis, the convergence criteria of the PSO algorithm was set at 0.05. Also, the number of particles present in this optimization problem was specified at 15. Lastly, the maximum number of iterations was set at 100.

Different optimization algorithms such as GA, DE, DA, and SLSQP were linked to the Bezier curve algorithm. This was done to determine the best optimizer for this optimization problem. Based on the results obtained, the DE algorithm proved to be the best optimizer in terms of average measured depth and standard deviation. However, further testing on the robustness and scalability of each algorithm must be conducted. In addition, to provide consistency with the optimizer used by the OGED team for well placement and facility optimization, it was decided that the PSO algorithm shall be used for this thesis study. However, all of the different algorithms were implemented into the integrated platform and can be used when needed for testing.

Based on the testing cases presented in this thesis, it was concluded that the algorithm (Bezier curve model and PSO optimizer) was able to produce an optimal solution when compared to the solution of the Bezier curve model (without the PSO optimizer). Furthermore, this improvement is more significant in the case of horizontal well targets. The results were visualized in Petrel to ensure that the trajectories obtained were feasible.

To further verify the strength of using the Bezier curve model, a comparison was done with the Well Path algorithm implemented in Petrel. This comparison showed that the Bezier curve algorithm was capable of generating shorter wells with a lower maximum DLS.

Once the results were validated, the algorithm was then implemented into the integrated well and facility placement optimization solution being developed by the OGED team. After implementation, the algorithm was used to successfully test different scenarios.

8.2. Recommendations for future work.

- Different optimization methods need to be tested.
- Implementation of other well trajectory models.

- Addition of geological constraints to the optimization problem.
- Modification of the algorithm to account for risks associated with drilling.
- Enhancing the process time of the optimziation algorithm being implemented in this thesis.

APPENDIX

```
# Bezier curves
import math
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from numpy import arctan2
import time

#from .node import Node
#from .well import Well
#from pso import PSO

# Parameters that can be changed:

# 1- KOP: Kick off point
# 2- distance_platform_cs: distance from the platform(well head) to the control point
Cs.
# 3- parametric_t: The t found in the parametric equation of the line to get the control
point Ce.
# 4- DLS_limit: The maximum DLS allowabl.

DLS_PARAMETER = 30
ERROR_THRESHOLD = 0.01
#global total_time
#total_time=0

# This cannot be changed
# 1- DLS_PARAMETER: is simply the 30 found in the equation of DLS. This is
inserted to obtain degrees/30 meters

# The function is divided to the following parts:
# Part 1: Enters the OGED file and identifies the given well. In this well, it selects the
drilling center and then removes it from the file to only have the horizontal well nodes.
# Part 2: The code chooses the intersection pt between well heel and well toe.
# Part 3: The vertical section of the well to reach KOP.
# Part 4: Azimuth of horizontal well and wellhead to target (this may be removed since
its not used like discussed)
# Part 5: Calculation of control points Cs and Ce. For Ce, I am using a parametric
equation.
# Part 6: Loop where t ranges from 0.01 to 1 to calculate coordinates and inclination
and azimuth.
# Part 7: Calculation of error between last point produced by the code and the point of
intersection(well heel or toe).
# Part 8: Add the horizontal well nodes to the produced trajectory to obtain a length of
120 nodes instead of 100.
```

```
# Part 9: Calculate the total depth or measured depth using eculidean distances.
# Part 10: Calculation of DLS for the whole trajectory (length of 120)
# Part 11: To check if in the length of the produced trajectory (length 100) DLS exceeds
DLS_limit of 7.
# Part 12: Loop to reach the maximum_parametric_t and see the first feasible trajectory.
# Part 13: Form a list to contain all outputs needed by Dr. Kassem.
# Part 14: Output the list in part 13 into a txt file and table format.


def read_well_data(filename):
    # To get the given horizontal well as an input:
    file=open(filename,'r')

    well_list=[]

    for line in file:
        splitLine=line.split(" ")
        x=float(splitLine[0])
        well_list.append(x)
        y=float(splitLine[1])
        well_list.append(y)
        z=float(splitLine[2])
        well_list.append(z)

    counter_list=len(well_list)
    if well_list[2]<0:
        platform_x=well_list[0]
        platform_y=well_list[1]
        platform_z=well_list[2]
        del well_list[0:3]
    else:
        platform_x=well_list[-3]
        platform_y=well_list[-2]
        platform_z=well_list[-1]
        del well_list[-3:]
    platform=[platform_x,platform_y,platform_z]
    well_array1=np.array([well_list])
    counter=well_array1.shape[1]
    a=counter/3
    well_array=well_array1.reshape(int(a),3)

    file.close()

    return well_array, platform

def bezier_curve(well_array, platform, KOP, distance_platform_cs, parametric_t):

    depth=platform[2]
```

```
z,gamma=0,0
trajectory_list=[]
increment=30
point=[]
initial_inclination=0
if len(well_array) == 2:
    final_inclination = 0
else:
    final_inclination = 90
initial_azimuth=0
direction_list=[]
inclination_list=[]
azimuth_list=[]
inclination_list,azimuth_list=[],[]

# To get target 1 and target 2
target1_array=well_array[0]
target1=target1_array.tolist()
target2_array=well_array[-1]
target2=target2_array.tolist()
platform_array=np.array([platform])
target_counter=well_array.shape[0]

# To choose the point of intersection with the given well.
distance_target1=np.linalg.norm(target1_array-platform_array)
distance_target2=np.linalg.norm(target2_array-platform_array)
if distance_target1<distance_target2:
        intersection_pt=target1
else:
    intersection_pt=target2

intersection_pt_array=np.array([intersection_pt])

# Vertical section of the well will end once the well reaches KOP.
while depth<KOP:
    point=[platform[0],platform[1],depth]
    depth+=increment
    trajectory_list.append(point)
    inclination_list.append(initial_inclination)
    azimuth_list.append(initial_azimuth)
point=[platform[0],platform[1],KOP]
inclination_list.append(initial_inclination)
azimuth_list.append(initial_azimuth)
trajectory_list.append(point)

# Calculate the azimuth of the kick off point and target
vector_platform_intersection_pt=[]
for element in range(len(intersection_pt)):
        x=intersection_pt[element]-point[element]
        vector_platform_intersection_pt.append(x)

    # Using trigonometric relations to guide the trajectory to target 1:
    azimuth_1=-
arctan2(vector_platform_intersection_pt[1],vector_platform_intersection_pt[0])+math.p
i/2
    if azimuth_1<0:
            azimuth_1=azimuth_1+2*math.pi
    maximum_azimuth1=math.degrees(azimuth_1)

    # Using trigonometric relations to get the azimuth of the horizontal well:
    horizontal_well_vector=[]
    if intersection_pt==target1:
        target1_n=well_array[1]
        target1_n_list=target1_n.tolist()
        for element in range(len(target1)):
            x=target1_n_list[element]-target1[element]
            horizontal_well_vector.append(x)
    else:
        for element in range(len(well_array)-1):
            continue
        target2_n=well_array[element]
        target2_n_list=target2_n.tolist()
        for element in range(len(target2)):
            x=target2[element]-target2_n[element]
            horizontal_well_vector.append(x)

    azimuth_2=-arctan2(horizontal_well_vector[1],horizontal_well_vector[0])+math.pi/2
    if azimuth_2<0:
            azimuth_2=azimuth_1+2*math.pi
    maximum_azimuth2=math.degrees(azimuth_2)

    # Calculate Ts and Te
    Ts,Te,Ti_list=[],[],[]
    Tsx,Tsy,Tsz,Tex,Tey,Tez=0,0,0,0,0,0


Tsx=(math.sin(math.radians(initial_inclination)))*(math.cos(math.radians(maximum_az
imuth1)))

Tsy=(math.sin(math.radians(initial_inclination)))*(math.sin(math.radians(maximum_az
imuth1)))
    Tsz=math.cos(math.radians(initial_inclination))
    Ts=[Tsx,Tsy,Tsz]
    Ts_array=np.array([Ts])
    Ti_list.append(Ts)
```

```
Tex=(math.sin(math.radians(final_inclination)))*(math.cos(math.radians(maximum_azi
muth2)))

Tey=(math.sin(math.radians(final_inclination)))*(math.sin(math.radians(maximum_azi
muth2)))
    Tez=math.cos(math.radians(final_inclination))
    Te=[Tex,Tey,Tez]
    Te_array=np.array([Te])


    # Calculate Cs and Ce
    Cex,Cey,Cez=0,0,0
    Cs,Ce=[],[]
    #de=(2.6232747288751776e+16)
    starting_pt_x,starting_pt_y,starting_pt_z=platform[0],platform[1],KOP
    starting_pt=[starting_pt_x,starting_pt_y,starting_pt_z]

intersection_pt_x,intersection_pt_y,intersection_pt_z=intersection_pt[0],intersection_pt
[1],intersection_pt[2]
    intersection_pt=[intersection_pt_x,intersection_pt_y,intersection_pt_z]

    Csx=starting_pt_x+(distance_platform_cs*Tsx)
    Csy=starting_pt_y+(distance_platform_cs*Tsy)
    Csz=starting_pt_z+(distance_platform_cs*Tsz)
    # Csx = starting_pt_x
    # Csy = starting_pt_y
    # Csz = starting_pt_z + distance_platform_cs
    Cs=[Csx,Csy,Csz]

    # To get Ce:
    if intersection_pt==target1:
        target_n1_array=well_array[1]
        target_n1=target_n1_array.tolist()
        direction_vector=target1_array-target_n1_array
        equation_line=(parametric_t*direction_vector)+target1_array
    else:
        target_n2_array=well_array[target_counter-2]
        target_n2=target_n2_array.tolist()
        direction_vector=target2_array-target_n2_array
        equation_line=(parametric_t*direction_vector)+target2_array
    Cex=equation_line[0]
    Cey=equation_line[1]
    Cez=equation_line[2]
    Ce=[Cex,Cey,Cez]

    # Calculate ti,inclination_i,azimuth_i
    u,counter=0,0
    Tix,Tiy,Tiz=0,0,0
    B_dot_x,B_dot_y,B_dot_z=0,0,0
    B_dot=[]
    inclination_i,azimuth_i=0,0
    list=[]
    while u<1:
        u+=0.01
        counter+=1
        B_dot_x=(-3*((1-u)**2)*starting_pt_x)+(3*(1-u)*(1-3*u)*Csx)+(3*(2-
3*u)*u*Cex)+(3*u*u*intersection_pt_x)
        B_dot_y=(-3*((1-u)**2)*starting_pt_y)+(3*(1-u)*(1-3*u)*Csy)+(3*(2-
3*u)*u*Cey)+(3*u*u*intersection_pt_y)
        B_dot_z=(-3*((1-u)**2)*starting_pt_z)+(3*(1-u)*(1-3*u)*Csz)+(3*(2-
3*u)*u*Cez)+(3*u*u*intersection_pt_z)
        B_dot=np.array([B_dot_x,B_dot_y,B_dot_z])


        B_dot_radical1=(B_dot*B_dot)**(0.5)
        B_dot_radical=np.linalg.norm(B_dot_radical1)

        Ti=(B_dot/B_dot_radical)
        Tix=Ti[0]
        Tiy=Ti[1]
        Tiz=Ti[2]
        Ti1=[Tix,Tiy,Tiz]
        Ti_list.append(Ti1)

        inclination_i=math.degrees(math.acos(Tiz))
        inclination_list.append(inclination_i)


        last_point_x=(((1-u)**3)*starting_pt_x)+(3*((1-u)**2)*u*Csx)+((3*(1-
u)*u*u*Cex))+((u**3)*intersection_pt_x)
        last_point_y=(((1-u)**3)*starting_pt_y)+(3*((1-u)**2)*u*Csy)+((3*(1-
u)*u*u*Cey))+((u**3)*intersection_pt_y)
        last_point_z=(((1-u)**3)*starting_pt_z)+(3*((1-u)**2)*u*Csz)+((3*(1-
u)*u*u*Cez))+((u**3)*intersection_pt_z)
        last_point=np.array([last_point_x,last_point_y,last_point_z])

        trajectory_list.append(last_point)

    error=np.linalg.norm(intersection_pt_array-last_point)

    if error < 0 or error > ERROR_THRESHOLD:
        trajectory_list = []
        return trajectory_list
    results = {
        'trajectory_list': trajectory_list,
        'well_array': well_array,
        'azimuth_list': azimuth_list,
        'inclination_list': inclination_list,
        'final_inclination': final_inclination,
        'intersection_pt_array': intersection_pt_array
    }

    return results
    #return
trajectory_list,well_array,azimuth_list,inclination_list,final_inclination,KOP,DLS_para
meter,DLS_limit,intersection_pt_array

# To add the horizontal well to the trajectory_list:
def append_horizontal_plannned(trajectory_list,well_array,intersection_pt_array):
    trajectory_list_length=len(trajectory_list)
    trajectory_list_modified=trajectory_list
    trajectory_array_plannedwell=np.array([trajectory_list])
    trajectory_array_plannedwell_counter=trajectory_array_plannedwell.shape[1]

    horizontal_well_entry_point1_array=np.array([well_array[0]])
    horizontal_well_entry_point2_array=np.array([well_array[-1]])

    well_horizontal_list=well_array.tolist()
    well_horizontal_list_length=len(well_horizontal_list)

    if np.linalg.norm(horizontal_well_entry_point1_array-intersection_pt_array)==0:
        del well_horizontal_list[0]
        for i in range(len(well_horizontal_list)):
            trajectory_list_modified.append(well_horizontal_list[i])
    elif np.linalg.norm(horizontal_well_entry_point2_array-intersection_pt_array)==0:
        del well_horizontal_list[-1]
        for i in reversed(range(len(well_horizontal_list))):
            trajectory_list_modified.append(well_horizontal_list[i])

    trajectory_array=np.array([trajectory_list_modified])
    trajectory_array_counter = trajectory_array.shape[1]

    results = {
        'trajectory_list_modified': trajectory_list_modified,
        'trajectory_array': trajectory_array,
        'trajectory_array_counter': trajectory_array_counter,
        'trajectory_array_plannedwell': trajectory_array_plannedwell,
        'trajectory_array_plannedwell_counter': trajectory_array_plannedwell_counter,
        'well_horizontal_list': well_horizontal_list
    }
    return results
    #return
well_horizontal_list,trajectory_list_modified,trajectory_array_counter,trajectory_array,t
rajectory_array_plannedwell,trajectory_array_plannedwell_counter

# To determine the position at which the KOP is found:
def azimuth_allpoints(trajectory_list,azimuth_list,KOP):

    azimuth_list_modified=azimuth_list
    for KOP_position in range(len(trajectory_list)):
        if trajectory_list[KOP_position][2]==KOP:
            break
        else:
            continue

    # To determine the azimuth of the points in the trajectory:
    delta_x,delta_y=0,0
    for i in range(KOP_position,len(trajectory_list)-1):
            delta_x=trajectory_list[i+1][0]-trajectory_list[i][0]
            delta_y=trajectory_list[i+1][1]-trajectory_list[i][1]
            azimuth=-arctan2(delta_y,delta_x)+math.pi/2
            if azimuth<0:
                azimuth=azimuth+2*math.pi
            azimuth_i=math.degrees(azimuth)
            azimuth_list_modified.append(azimuth_i)

    return azimuth_list_modified

# To calculate the euclidean distance of all the points in the well:
def
Measured_distance(trajectory_list_modified,trajectory_array_plannedwell,trajectory_ar
ray):
    MD_list=[]
    distance_all=[]
    trajectory_array_counter = len(trajectory_array[0])
    trajectory_array_plannedwell_counter = len(trajectory_array_plannedwell[0])
    for i in range(trajectory_array_counter-1):
        distance=np.linalg.norm(trajectory_array[0][i+1]-trajectory_array[0][i])
        distance_all.append(distance)
    MD=0
    for i in range(trajectory_array_plannedwell_counter-1):
        MD+=distance_all[i]
        MD_list.append(MD)

    results = {
        'distance_all': distance_all,
        'MD_list': MD_list,
        'MD': MD,
        'trajectory_array': trajectory_array
    }

    return results

# To calculate DLS for distance_all:
def
DLS_all(well_horizontal_list,inclination_list,final_inclination,azimuth_list_modified,di
stance_all,DLS_parameter):
    inclination_list_modified=inclination_list
    for i in range(len(well_horizontal_list)):
        inclination_list_modified.append(final_inclination)

    inclination_array_counter=len(inclination_list_modified)

inclination_array=np.array([inclination_list_modified]).reshape(inclination_array_count
er,1)

azimuth_array=np.array([azimuth_list_modified]).reshape(inclination_array_counter,1)

    dogleg_angle_list=[]
    for i in range(inclination_array_counter-1):

dogleg_angle=math.acos(math.cos(math.radians(inclination_array[i]))*math.cos(math.r
adians(inclination_array[i+1]))+math.sin(math.radians(inclination_array[i]))*math.sin(
math.radians(inclination_array[i+1]))*math.cos(math.radians(azimuth_array[i+1]-
azimuth_array[i])))
        dogleg_angle_list.append(math.degrees(dogleg_angle))


dogleg_angle_array=np.array([dogleg_angle_list]).reshape(inclination_array_counter-
1,1)

    DLS_list=[]
    for i in range(inclination_array_counter-1):
        if distance_all[i]==0:
            DLS=0
            DLS_list.append(DLS)
        else:
            DLS=(DLS_parameter/distance_all[i])*dogleg_angle_array[i][0]
            DLS_list.append(DLS)

    results = {
        'DLS_list': DLS_list,
        'inclination_list_modified': inclination_list_modified,
        'dogleg_angle_list': dogleg_angle_list
    }
    return results

# To caclulate DLS untill intersection_pt is reached:
def
DLS_planned_trajectory(DLS_list,trajectory_array_plannedwell,DLS_limit,trajectory_l
ist_modified):

    DLS_planned_well_list=[]

    for i in range(len(trajectory_array_plannedwell[0])-1):
        DLS_planned_well_list.append(DLS_list[i])

    for i in DLS_planned_well_list:
        if i>DLS_limit:
            trajectory_list_modified=[]
            break

    result = {
        'DLS_planned_well_list': DLS_planned_well_list,
        'trajectory_list_modified': trajectory_list_modified
    }

    return result


class WellTrajetoryDesign:

    def__init_(self, well=None, platform=None, well_heel=0, DLS_limit=7,
KOP=600):
        self.well_nodes = well
        self.platform = platform
        self.well_heel = well_heel
        self.DLS_limit = DLS_limit
        self.KOP = KOP

        well_nodes = self.well_nodes

        if self.well_heel == 0:
            self.well_array = [ node for node in well_nodes ]
        else:
            self.well_array = [ node for node in reversed(well_nodes) ]

        self.well_array = np.array(self.well_array)

        return

    def get_well_trajectory(self, well_array, platform, distance_platform_cs,
parametric_t):
        solution = bezier_curve(well_array, platform, self.KOP, distance_platform_cs,
parametric_t)

        if not solution: return None #If failed to construct bezier curve, return

        trajectory_list = solution['trajectory_list']
        intersection_pt_array = solution['intersection_pt_array']
        azimuth_list = solution['azimuth_list']
        inclination_list = solution['inclination_list']
        final_inclination = solution['final_inclination']

        solution = append_horizontal_plannned(trajectory_list, well_array,
intersection_pt_array)
        trajectory_list_modified = solution['trajectory_list_modified']
        trajectory_array = solution['trajectory_array']
        trajectory_array_counter = solution['trajectory_array_counter']
        trajectory_array_plannedwell = solution['trajectory_array_plannedwell']
        trajectory_array_plannedwell_counter =
solution['trajectory_array_plannedwell_counter']
        well_horizontal_list = solution['well_horizontal_list']

        azimuth_list_modified =
azimuth_allpoints(trajectory_list_modified,azimuth_list,self.KOP)

        solution = Measured_distance(trajectory_list_modified,
trajectory_array_plannedwell, trajectory_array)
        distance_all = solution['distance_all']
        MD_list = solution['MD_list']
        MD = solution['MD']
        trajectory_array = solution['trajectory_array']


        solution = DLS_all(well_horizontal_list, inclination_list,
final_inclination,azimuth_list_modified ,distance_all, DLS_PARAMETER)
        DLS_list = solution['DLS_list']
        inclination_list_modified = solution['inclination_list_modified']
        dogleg_angle_list = solution['dogleg_angle_list']

        solution = DLS_planned_trajectory(DLS_list,trajectory_array_plannedwell,
self.DLS_limit, trajectory_list_modified)
        DLS_planned_well_list = solution['DLS_planned_well_list']
        trajectory_list_modified = solution['trajectory_list_modified']

        if not trajectory_list_modified: return None #If exceeded DLS limit, return

        result = {
            'well_trajectory': trajectory_list_modified,
            'azimuth_list': azimuth_list_modified,
            'inclination_list': inclination_list_modified,
            'DLS_list': DLS_list,
            'MD_list': MD_list,
            'intersection_pt_counter': trajectory_array_plannedwell_counter,
            'trajectory_array_counter': trajectory_array_counter
        }

        return result

    def generate_well_trajectory(self, well, heel_index, platform, distance_platform_cs,
parametric_t):

        well_nodes = well.trajectory_nodes
        well_array = []

        if heel_index == 0:
            well_array = [ [node.x, node.y, node.prop] for node in well_nodes ]
        else:
            well_array = [ [node.x, node.y, node.prop] for node in reversed(well_nodes) ]

        well_trajectory_solution = self.get_well_trajectory(well_array, platform,
distance_platform_cs, parametric_t)

        well_nodes = [ Node(n[0], n[1], n[2]) for n in
well_trajectory_solution['well_trajectory'][0]]
        well.trajectory_nodes = well_nodes
        well.well_trajectory_solution = well_trajectory_solution

        return well

    def cost_function(self, positions, particle_id, close_to_convergence=False):
        distance_platform_cs = positions[0]
        parametric_t = positions[1]
        KOP = positions[2]

        self.KOP = KOP

        well_trajectory_solution = self.get_well_trajectory(self.well_array, self.platform,
distance_platform_cs, parametric_t)

        if not well_trajectory_solution:
            return 1e4

        return well_trajectory_solution['MD_list'][-1]

    def well_trajectory_final_iteration(self, positions):
        distance_platform_cs = positions[0]
        parametric_t = positions[1]
        KOP = positions[2]

        self.KOP = KOP

        well_trajectory_solution = self.get_well_trajectory(self.well_array, self.platform,
distance_platform_cs, parametric_t)

        if not well_trajectory_solution:
            return None

        return well_trajectory_solution


def optimize_well_trajectory_PSO(well_nodes, platform, well_heel=0, DLS_Limit=7):
    platform = np.array(platform)
    well_nodes = np.array(well_nodes)
    well_trajectory_design = WellTrajetoryDesign(well_nodes, platform,
DLS_limit=DLS_Limit)

    bounds = [
            (0, 1000),
            (0, 100),
            (200, 600)
        ]

    number_particles = 15
    number_iterations = 100
    convergence_threshold = 0.05

    initial = []
    #init0,init1,init2=0,0,0
    for i in range(number_particles):
        init = []
        #init =np.array([init0,init1,init2])
        #init[0]=np.random.uniform(300,900)
        #init[1]=np.random.uniform(0,2)
        #init[2]=np.random.uniform(200,400)
        for j, bound in enumerate(bounds):
            init.append(np.random.uniform(bound[0], bound[1]))

        initial.append(init)

    s = time.time()
    pso = PSO(well_trajectory_design.cost_function, initial, bounds, number_particles,
number_iterations, convergence=convergence_threshold, is_max=False,
run_parallel=False, plot_err=True)
    pso.run_optimization_sync()
    e = time.time()
    #total_time+=(e-s)

    best_particle_position = pso.pos_best_g
    best_npv = pso.err_best_g

    print(best_particle_position)
    print(best_npv)
    print(e-s)

    well_trajectory_solution =
well_trajectory_design.well_trajectory_final_iteration(best_particle_position)
    if well_trajectory_solution is None :
        return None
    trajectory_list = well_trajectory_solution['well_trajectory']


    print("Done")

    return well_trajectory_solution



def optimize_well_trajectory(well_nodes, platform, well_heel=0, DLS_Limit=7,
KOP=600):

    well_trajectory_design = WellTrajetoryDesign(DLS_limit=DLS_Limit, KOP=KOP)

    if well_heel == 0:
        well_array = np.array([ [node.x, node.y, node.prop] for node in well_nodes[:] ])
    else:
        well_array = np.array([ [node.x, node.y, node.prop] for node in
reversed(well_nodes[:]) ])

    platform = np.array([platform.x, platform.y, platform.prop])

    #Very simple un-optimized 'optimizer'
    maximum_parametric_t=0
    distance_platform_cs=0

    done = False
    trajectory_list_solution = None



    '''

    while maximum_parametric_t<200 and not done:
        maximum_parametric_t+=1

        while distance_platform_cs < 1000 and not done:

            distance_platform_cs += 200
            distance_platform_cs = 600
            trajectory_list_solution = well_trajectory_design.get_well_trajectory(well_array, 
platform, 
distance_platform_cs=distance_platform_cs,parametric_t=maximum_parametric_t)

            if not trajectory_list_solution:
                break#continue
            else:
                done = True
                break

    return trajectory_list_solution
    '''
```

REFERENCES

Helmy, M. W., Khalaf, F., & Darwish, T. A. (1998). Well Design Using a Computer Model. *Society of Petroleum Engineers*.

Allain, O., Dyson, M., Jing, X., Pentland , C., Polikar, M., & Suicmez, V. S. (2019). The Imperial College Lectures in Petroleum Engineering Drilling and Reservoir Appraisal. *World Scientific Publishing Europe Ltd.*

Amorin, R., & Bediako, E. B. (2010). Application of Minimum Curvature Method to Wellpath Calculations. *Research Journal of Applied Sciences, Engineering and Technology*.

Amorin, R., & Broni-Bediako, E. (2010). Application of Minimum Curvature Method to Wellpath Calculations. *Research Journal of Applied Sciences, Engineering and Technology*.

ARABJAMALOEI, R., EDALATKHAH, S., & JAMSHIDI, E. (2011). A New Approach to Well Trajectory Optimization Based on Rate of Penetration and Wellbore Stability. *Petroleum Science and Technology*.

Atashnezhad, A., Wood, D. A., Fereidounpour, A., & Khosravanian, R. (2014). Designing and optimizing deviated wellbore trajectories using novel particle swarm algorithms. *Journal of Natural Gas Science and Engineering*.

Azar, J. J., & Samuel, G. R. (2007). *Drilling Engineering.* PennWell Corporation.

Biswas, K., Islam, T., Joy, J. S., Vasant, P., Vintaned, J. A., & Watada, J. (2020). Multi-Objective Spotted Hyena Optimizer for 3D Well Path Optimization. *Journal of Science and Sustainable Development*.

Bourgoyne Jr., A. T., Millheim, K. K., Chenevert, M. E., & Young Jr., F. (1991). *Applied Drilling Engineering.* Society of Petroleum Engineers.

Denney, D. (2011). Overcoming Challenges of Drilling High-Dogleg-Severity Curves. *Journal of Petroleum Technology*.

Devereux, S. (1998). *Practical Well Planning and Drilling Manual.* PennWell Corporation.

Farah, F. O. (2013). *DIRECTIONAL WELL DESIGN, TRAJECTORY AND SURVEY CALCULATIONS, WITH A CASE STUDY IN FIALE, ASAL RIFT, DJIBOUTI.* United Nations University.

Guo, B., Miska, S., & Lee, R. L. (1992). Constant Curvature Method for Planning a 3-D Directional Well. *Society of Petroleum Engineers*.

Guria, C., Goli, K. K., & Pathak , A. K. (2014). Multi-objective optimization of oil well drilling using elitist non-dominated sorting genetic algorithm. *Springer*.

Halafawi, M., & Avram, L. (February 05, 2019). Wellbore Trajectory Optimization for Horizontal Wells: The Plan Versus the Reality. *Journal of Oil, Gas and Petrochemical Sciences*.

Harb, A., Kassem, H., & Ghorayeb, K. (2019). Black hole particle swarm optimization for well placement optimization. *Springer Nature Switzerland*.

Hossain, M. E., & Al-Majed, A. A. (2015). *Fundamentals of Sustainable Drilling Engineering.* Scrivener Publishing.

Hosseini, S., Ghanbarzadeh, A., & Hashemi, A. (2014). *Optimization of Dogleg Severity in Directional Drilling Oil Wells Using Particle Swarm Algorithm.* Journal of Chemical and Petroleum Engineering.

Inglis, T. A. (1987). *Directional Drilling.* Graham &Trotman Limited.

Jones, R., Rose, J., Laurie, P., Hibbert, E., Butler, P., Freeman, A., . . . Halliburton Energy Services. (1997). *Design, Planning, Implementation & Management of a Multi-Lateral Well on the BP Forties FIeld: A North Sea Case History.* Soceity of Petroleum Engineers.

Joshi, D. R., & Samuel, R. (2017). Automated Geometric Path Correction in Directional Drilling. *Society of Petroleum Engineers*.

Konstantakopoulos, I. K., & Stamataki, S. K. (1996). A Computerized Method for the Real-Time Well Path Monitoring and Placement in 3-D Space. *Society of Petroleum Engineers*.

Kraft, D. (1988). A Software Package for Sequential Quadratic Programming.

Krishnan, A., & Kulkarni, A. (2016). Well Trajectory Survey of a Directional well. *International Research Journal of Engineering and Technology (IRJET)*.

Ma, T., Chen, P., & Zhao, J. (2016). Overview on Vertical and Directional Drilling Technologies for the Exploaration and Explotation of Deep Petroleum Resources. *Springer International Publishing Switzerland*.

Ma, T., Chen, P., Yang, C., & Zhao, J. (2015). Wellbore stability analysis and well path optimization based on the breakout width model and Mogi–Coulomb criterion. *Journal of Petroleum Science and Engineering*.

Mansouri, V., Khosravanian, R., Wood, D. A., & Aadnoy, B. S. (2015). 3-D well path design using a multi objective genetic algorithm. *Journal of Natural Gas Science and Engineering*.

McClendon, R. T., & Anders, E. O. (1985). Directional Drilling Using the Catenary Method. *Society of Petroleum Engineers*.

McMillian, W. H. (1980). Planning the Directional Well - A Calculation Method. *Journal of Petroleum Technology* .

Mirjalili, S. (2018). *Evolutionary Algorithms and Neural Networks Theory and Applications.*

Mitchell, B. (1995). *Advanced Oil Well Drilling Engineering Handbook and Computer Programs.* The Society of Petroleum Engineers of the AIME.

Navarro, A. R., & Yalcin, O. (1990). Determination of the Drillability of High-Angle, Large-Displacement Wells by Analyzing the Effects of Dog-Leg Severity. *Society of Petroleum Engineers*.

Okpozo, P. O., Peters, A., & Okologume, W. C. (2016). Directional Well Trajectory Design: The Effect of Change of Azimuth Builds and Turns On Build-&-Hold and Continuous Build Trajectories. *Journal of the Nigerian Association of Mathematical Physics*.

Price, K. V., Storn, R. M., & Lampinen, J. A. (2005). *Differential Evolution A Practical Approach to Global Optimization.* Springer.

Salman, M., Joshi, D., & Samuel, R. (2017). Application of Geomechanics and Hybrid Metaheuristics in Designing an Optimized Well Path. *Society of Petroleum Engineers*.

Sampaio Jr, J. H. (2016). Designing 3D Directional Well Trajectories Using Bezier Curves. *Journal of Energy Resources Technology*.

Sampaio, J. H. (2006). Planning 3D Well Trajectories Using Cubic Functions. *Journal of Energy Resources Technology*.

Sampaio, J. H. (2007). Planning 3D Well Trajectories Using Spline-in-Tension Functions. *Journal of Energy Resources Technology*.

Sawaryn, S. J., & Thorogood, J. L. (2003). A Compendium of Directional Calculations Based on the Minimum Curvature Method. *Society of Petroleum Engineers*.

Scholes, H. (1983). A Three-Dimensional Well Planning Method for HDR. *Society of Petroleum Engineers*.

Schulze-Riegert, R., Bagheri, M., Krosche , M., Kück, N., & Ma, D. (2011). Multiple-Objective Optimization Applied to Well Path Design under Geological Uncertainty. *Society of Petroleum Engineers*.

SCM. (2009). *Petrel help manual.*

Seyyedattar, S. M., Parvizi, H., & Malekzadeh, E. (2012). Optimization of Wellhead Location Selection and Well Path Design. *Petroleum Science and Technology*.

Sha, L., & Pan, Z. (2018). FSQGA based 3D complexity wellbore trajectory optimization. *Oil & Gas Science and Technology*.

Shokir, E. E., Emera, M. K., Eid, S. M., & Wally, A. W. (2004). A New Optimization Model for 3D Well Design. *Oil & Gas Science and Technology*.

Short, J. (1993). *Introduction to Directional and Horizontal Drilling.* PennWell Publishing Company.

Wang, Z., & Inglis, T. A. (December 1990). Planning Directional Wells Through a. *SPE Drilling Engineering*.

Wang, Z., Gao, D., & Yang, J. (2019). Desing and Calculation of Complex Directional-Well Trajectories on the Basis of the Minimum-Curvature Method. *SPE Drilling & Completion*.

Wiśniowski, R., Łopata, P., & Orłowicz, G. (2020). Numerical Methods for Optimization of the Horizontal Directional Drilling (HDD) Well Path Trajectory. *Energies*.

Wood, D. A. (2016). Hybrid bat flight optimization algorithm applied to complex wellbore trajectories highlights the relative contributions of metaheuristic components. *Journal of Natural Gas Science and Engineering*.

Wood, D. A. (2016). Hybrid cuckoo search optimization algorithms applied to complex wellbore trajectories aided by dynamic, chaos-enhanced, fat-tailed distribution sampling and metaheuristic profiling. *Journal of Natural Gas Science and Engineering*.

Xiang, Y., Gubian, S., Suomela, B., & Hoeng, J. (2013). Generalized Simulated Annealing for Global Optimization: The GenSA Package. *The R Journal*.