Exercis One

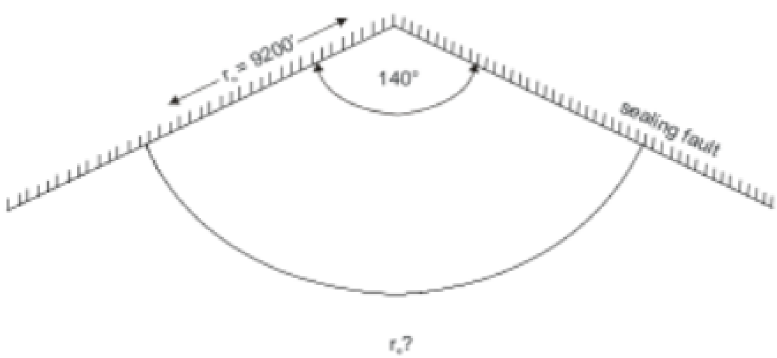
January 15, 2021

1 Applicant Details

- Full Name: Omeh, Chukwuemeka Christian
- Email: chukwuemekao241@gmail.com
- Phone Number: 08165468858
- Linkeld Profile: Click Here
- Assessment Github Repo: Click Here

2 Exercise One

Exercise One (From L.P. Dake Exercise 9.2)
A wedge-shaped reservoir is suspected of having a fair strong natural water drive. The geometry of the reservoir-aquifer system is shown below. The reservoir was at bubble point at the initial condition with no gas cap (m=0). Develop a computer program that will Perform history matching of the reservoir using the production and PVT data given below.
Hint: Use Klins et al paper: "A Polynomial approach to the Van Everdingen - Hurst dimensionless variables for water encroachment"



Reservoir properties

$N = 312 \times 10^6$ stb
$S_{wc} = 0.05$

Properties common to the reservoir and aquifer

h = 100 ft
k = 200 mD
μ_w = 0.55 cp
ϕ = 0.25
c_w = 3.0×10^{-6} /psi
c_f = 4.0×10^{-6} / psi
B_w = 1.0

2.1 PROJECT ARCHITECTURE

2.1.1 Tools

- Excel
- Python language
- Pycharm IDE
- Flask
- Turtle
- Tkinter
- Jupyter Notebook
- Git and Github

2.1.2 Key points

- Fair strong natural water drive mechanism.
- Wedge-shaped reservoir.
- Initial Conditions are at bubble point(P and T).
- No gas cap(m = 0).

2.1.3 Aim

- To develop a computer program that will perform history matchingof the reservoir using the production and PVT data given.

2.1.4 PVT Data:

Time Pressure at the Plateau Pressure Δp

Time (years)	N_p (MM/stb)	R_p (scf/stb)	B_o (rb/stb)	R_s (scf/stb)	B_g (rb/scf)
0		650 (R_{si})	1.404(B_{oi})	650 (R_{si})	.00093 (B_{gi})
1	7.88	760	1.374	592	.00098
2	18.42	845	1.349	545	.00107
3	29.15	920	1.329	507	.00117
4	40.69	975	1.316	471	.00128
5	50.14	1025	1.303	442	.00139
6	58.42	1065	1.294	418	.00150
7	65.39	1095	1.287	398	.00160
8	70.74	1120	1.280	383	.00170
9	74.54	1145	1.276	381	.00176
10	77.43	1160	1.273	364	.00182

2.1.5 Production Data

2.1.6 Approach

- Using Klins et al paper: "A Polynomial approach to the Van Everdingen - Hurst dimensionless variables for water encroachment"

1. The van Everdingen and Hurst dimensionless variables rho/sub D/ and q/sub D/.

3 Importing Necessary Python Libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy
```

4 Setting Result Display Margins

[2]:
```
pd.set_option("display.max_column", None)
pd.set_option("display.max_row", None)
pd.set_option("display.max_colwidth", None)
#pd.set_option("display.float_format", lambda x: "%.2f" %x)
plt.style.use("ggplot")
```

[3]:
```
# Creating PVT Data table
"""
Table Features
1. Time(yrs)
2. Pressure at Oil water contact(psia)
3. Plateau Pressure Level(psia)
4. Change in pressure(psia)
5. Cummulative Oil Production, Np(MM/stb)
6. Cummulative Gas Oil Ratio, GOR(scf/stb)
7. Oil formation volume factor,Bo(rb/stb)
8. Gas Solubility, Rs(scf/stb)
9. Gas Formation Volume Factor, Bg(rb/scf)

The first three(1-3) are independent Variable while 4-9 are dependent variables␣
↪which can be
empirically determined using there respective equations/correlations.

Though, the values of the dependent variables are already given in the PVT␣
↪table shown, we will
try to automate its determination and append the results into the table.

"""
```

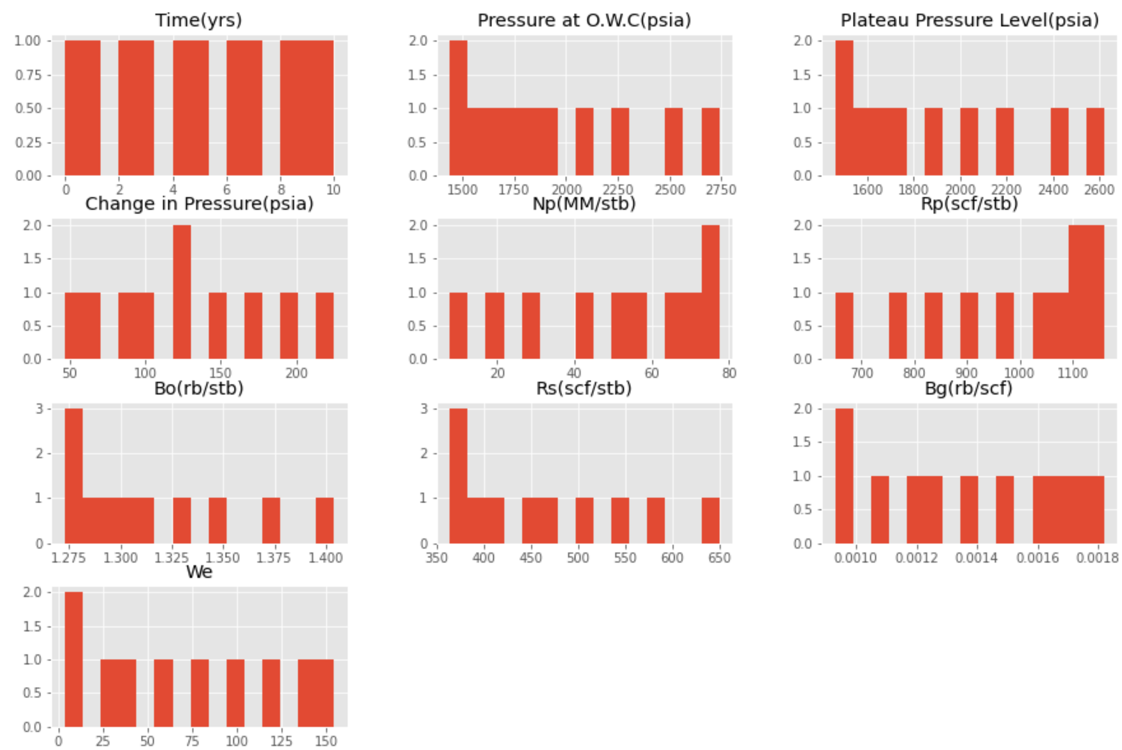
```python
pvt_table = {"Time(yrs)": [* range(11)],
             "Pressure at O.W.C(psia)": [2740, 2500, 2290, 2109, 1949, 1818,␣
↪1702, 1608, 1635, 1480, 1440],
             "Plateau Pressure Level(psia)": [np.nan, 2620, 2395, 2199, 2029,␣
↪1883, 1760, 1655, 1571, 1507, 1460],
             "Change in Pressure(psia)": [120, 225, 196, 170, 146, 123, 105,␣
↪84, 64, 47, np.nan],
             "Np(MM/stb)": [np.nan, 7.88, 18.42, 29.15, 40.69, 50.14, 58.42, 65.
↪39, 70.74, 74.54, 77.43],
             "Rp(scf/stb)": [650, 760, 845, 920, 975, 1025, 1065, 1095, 1120,␣
↪1145, 1160],
             "Bo(rb/stb)": [1.404, 1.374, 1.349, 1.329, 1.316, 1.303, 1.294, 1.
↪287, 1.280, 1.276, 1.273],
             "Rs(scf/stb)": [650, 592, 545, 507, 471, 442, 418, 398, 383, 381,␣
↪364],
             "Bg(rb/scf)": [0.00093, 0.00098, 0.00107, 0.00117, 0.00128, 0.
↪00139, 0.00150, 0.00160, 0.00170,
                            0.00176, 0.00182],
             "We": [np.nan, 3.829, 13.460, 26.462, 41.935, 59.207, 77.628, 96.
↪805, 116.284, 135.601, 154.401]
            }

pvt_data = pd.DataFrame.from_dict(pvt_table)
pvt_data.head(11)
```

[3]:

```
    Time(yrs)  Pressure at O.W.C(psia)  Plateau Pressure Level(psia)  \
0           0                     2740                           NaN
1           1                     2500                        2620.0
2           2                     2290                        2395.0
3           3                     2109                        2199.0
4           4                     1949                        2029.0
5           5                     1818                        1883.0
6           6                     1702                        1760.0
7           7                     1608                        1655.0
8           8                     1635                        1571.0
9           9                     1480                        1507.0
10         10                     1440                        1460.0

    Change in Pressure(psia)  Np(MM/stb)  Rp(scf/stb)  Bo(rb/stb)  \
0                      120.0         NaN          650       1.404
1                      225.0        7.88          760       1.374
2                      196.0       18.42          845       1.349
3                      170.0       29.15          920       1.329
4                      146.0       40.69          975       1.316
5                      123.0       50.14         1025       1.303
```

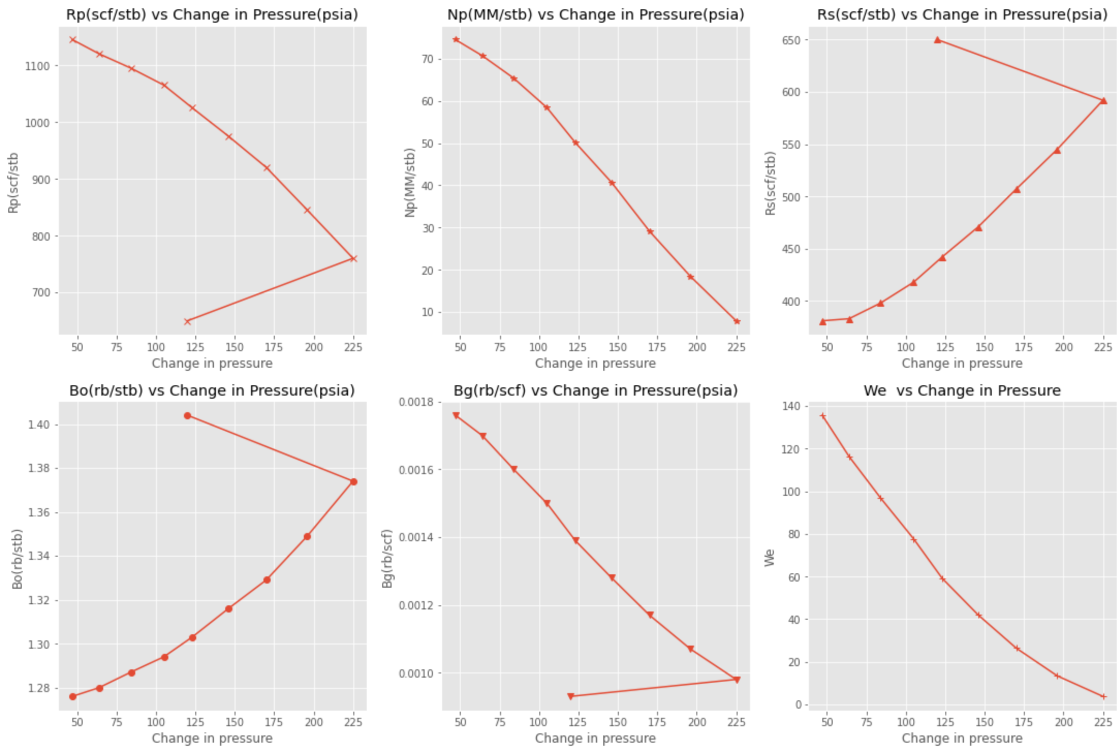
```
6                        105.0     58.42          1065        1.294
7                         84.0     65.39          1095        1.287
8                         64.0     70.74          1120        1.280
9                         47.0     74.54          1145        1.276
10                         NaN     77.43          1160        1.273

    Rs(scf/stb)  Bg(rb/scf)       We
0           650     0.00093      NaN
1           592     0.00098    3.829
2           545     0.00107   13.460
3           507     0.00117   26.462
4           471     0.00128   41.935
5           442     0.00139   59.207
6           418     0.00150   77.628
7           398     0.00160   96.805
8           383     0.00170  116.284
9           381     0.00176  135.601
10          364     0.00182  154.401
```

[4]:
```
pvt_data.hist(bins = 15, figsize = (15,10));
```



[5]:
```
f, ax = plt.subplots(nrows=2, ncols=3, figsize=(15,10), constrained_layout=True)
#f.tight_layout()
```

```
ax[0][0].plot(pvt_data["Change in Pressure(psia)"], pvt_data["Rp(scf/stb)"],␣
↪marker = "x")
ax[0][0].set_title("Rp(scf/stb) vs Change in Pressure(psia)")
ax[0][0].set(xlabel='Change in pressure', ylabel='Rp(scf/stb')
ax[0][1].plot(pvt_data["Change in Pressure(psia)"], pvt_data["Np(MM/stb)"],␣
↪marker = '*')
ax[0][1].set_title("Np(MM/stb) vs Change in Pressure(psia)")
ax[0][1].set(xlabel='Change in pressure', ylabel='Np(MM/stb)')
ax[0][2].plot(pvt_data["Change in Pressure(psia)"], pvt_data["Rs(scf/stb)"],␣
↪marker = '^')
ax[0][2].set_title("Rs(scf/stb) vs Change in Pressure(psia)")
ax[0][2].set(xlabel='Change in pressure', ylabel='Rs(scf/stb)')
ax[1][0].plot(pvt_data["Change in Pressure(psia)"], pvt_data["Bo(rb/stb)"],␣
↪marker = "o")
ax[1][0].set_title("Bo(rb/stb) vs Change in Pressure(psia)")
ax[1][0].set(xlabel='Change in pressure', ylabel='Bo(rb/stb)')
ax[1][1].plot(pvt_data["Change in Pressure(psia)"], pvt_data["Bg(rb/scf)"],␣
↪marker = "v")
ax[1][1].set_title("Bg(rb/scf) vs Change in Pressure(psia)")
ax[1][1].set(xlabel='Change in pressure', ylabel='Bg(rb/scf)')
ax[1][2].plot(pvt_data["Change in Pressure(psia)"],pvt_data["We"], marker = "+")
ax[1][2].set_title("We  vs Change in Pressure")
ax[1][2].set(xlabel='Change in pressure', ylabel='We')
plt.show()
```



- The above table represent the production data over time.
- The above table will serve as our observed data or what is usually refered to production history data
- We shall perform history matching by making a comparison between the observed history data and the calculated which is from our Algorithm using Klins et al paper: ”Polynomial Approach to the Van Everdigen - Hurst Dimensionless variables for water encroachment. The paper is basically using numerical approxiamation.
- let Calculate the Cumulative water influx using the given production and PVT data
- Since the value of rD wasn't give, we will run a for loop from range 2 - 25 as stated in Klin et al paper

[6]:
```
# Using constant terminal rate, pD
# Determining tD using rD of 10

def Klins_params_verification(
k = int(input("Enter value of permeability in mD")),
t = int(input("Enter the maximum value of in yrs")),
phi = float(input("Enter the phase angle value phi")),
Uw = float(input("Enter the water viscosity value")),
Cw = float(input("Enter the water compressibility value, Uw")),
Cf = float(input("Enter the formation compressibility value, Cf")),
ro = float(input("Enter the reservoir radius value, ro"))):

    Cfw = Cw + Cf


    tD1 = []
    t_cross1 = []
    for time in range(t + 1):
        tD = (2.309 * k * time) / (phi * Uw * Cfw * pow(ro, 2))
        print()
        print("=====================RESULTS=====================")
        print("At time(t):",time, "\t","The dimensionless time is:", round(tD,␣
 ↪2))
        tD1.append(tD)

    print()
   ␣
 ↪print("==================================================================================="
    print("Let Calculate the crossover time to know if the aquifer is finite or␣
 ↪infinite")
   ␣
 ↪print("==================================================================================="
```

```
# Calculating crossover time to ascertain if is finite or infinite aquifer
    for rD in range(2, 26):
        bo = 0.0980958
        b1 = 0.100683
        b2 = 2.03863
        t_cross = bo * (rD - 1) + b1 * (rD - 1) ** b2
        print()
        ␣
↪print("======================RESULTS==================================")
        print("The dimensionless radius is:",rD, "\t", "The crossover time␣
↪corresponding to tD is:", round(t_cross, 2))
        t_cross1.append(t_cross)

    print("=====TESTING IF AQUIFER IS FINITE OR INFINITE=====")

    for (val1, val2) in zip(tD1, t_cross1):
        if val1 >= val2:
            print("===Is a finite Aquifer===")
        else:
            print("===Is an infinite Aquifer===")
    pass

    """
    The results shows we have a finite aquifer.
    """

    print(f"The dimensionless time values are:,\n{tD1}", "\n")
    print("=============END===============")
    print(f"The crossover times are:, \n{t_cross1}", "\n")
    print("=============END================")
    print("==The results shows a finite aquifer===")

params = Klins_params_verification()
params
```

```
Enter value of permeability in mD200
Enter the maximum value of in yrs10
Enter the phase angle value phi0.25
Enter the water viscosity value0.55
Enter the water compressibility value, Uw0.000003
Enter the formation compressibility value, Cf0.000004
Enter the reservoir radius value, ro9200

======================RESULTS=====================
At time(t): 0    The dimensionless time is: 0.0

======================RESULTS=====================
```

```
At time(t): 1    The dimensionless time is: 5.67

======================RESULTS======================
At time(t): 2    The dimensionless time is: 11.34

======================RESULTS======================
At time(t): 3    The dimensionless time is: 17.01

======================RESULTS======================
At time(t): 4    The dimensionless time is: 22.67

======================RESULTS======================
At time(t): 5    The dimensionless time is: 28.34

======================RESULTS======================
At time(t): 6    The dimensionless time is: 34.01

======================RESULTS======================
At time(t): 7    The dimensionless time is: 39.68

======================RESULTS======================
At time(t): 8    The dimensionless time is: 45.35

======================RESULTS======================
At time(t): 9    The dimensionless time is: 51.02

======================RESULTS======================
At time(t): 10   The dimensionless time is: 56.69

================================================================================
===
Let Calculate the crossover time to know if the aquifer is finite or infinite
================================================================================
===

=======================RESULTS==================================
The dimensionless radius is: 2   The crossover time corresponding to tD is: 0.2

=======================RESULTS==================================
The dimensionless radius is: 3   The crossover time corresponding to tD is: 0.61

=======================RESULTS==================================
The dimensionless radius is: 4   The crossover time corresponding to tD is: 1.24

=======================RESULTS==================================
The dimensionless radius is: 5   The crossover time corresponding to tD is: 2.09

=======================RESULTS==================================
```

```
The dimensionless radius is: 6   The crossover time corresponding to tD is: 3.17

=======================RESULTS==================================
The dimensionless radius is: 7   The crossover time corresponding to tD is: 4.47

=======================RESULTS==================================
The dimensionless radius is: 8   The crossover time corresponding to tD is: 6.01

=======================RESULTS==================================
The dimensionless radius is: 9   The crossover time corresponding to tD is: 7.77

=======================RESULTS==================================
The dimensionless radius is: 10          The crossover time corresponding to tD
is: 9.76

=======================RESULTS==================================
The dimensionless radius is: 11          The crossover time corresponding to tD
is: 11.99

=======================RESULTS==================================
The dimensionless radius is: 12          The crossover time corresponding to tD
is: 14.44

=======================RESULTS==================================
The dimensionless radius is: 13          The crossover time corresponding to tD
is: 17.14

=======================RESULTS==================================
The dimensionless radius is: 14          The crossover time corresponding to tD
is: 20.06

=======================RESULTS==================================
The dimensionless radius is: 15          The crossover time corresponding to tD
is: 23.23

=======================RESULTS==================================
The dimensionless radius is: 16          The crossover time corresponding to tD
is: 26.62

=======================RESULTS==================================
The dimensionless radius is: 17          The crossover time corresponding to tD
is: 30.26

=======================RESULTS==================================
The dimensionless radius is: 18          The crossover time corresponding to tD
is: 34.13

=======================RESULTS==================================
```

```
The dimensionless radius is: 19         The crossover time corresponding to tD
is: 38.24

=======================RESULTS==================================
The dimensionless radius is: 20         The crossover time corresponding to tD
is: 42.59

=======================RESULTS==================================
The dimensionless radius is: 21         The crossover time corresponding to tD
is: 47.18

=======================RESULTS==================================
The dimensionless radius is: 22         The crossover time corresponding to tD
is: 52.0

=======================RESULTS==================================
The dimensionless radius is: 23         The crossover time corresponding to tD
is: 57.07

=======================RESULTS==================================
The dimensionless radius is: 24         The crossover time corresponding to tD
is: 62.38

=======================RESULTS==================================
The dimensionless radius is: 25         The crossover time corresponding to tD
is: 67.92
=====TESTING IF AQUIFER IS FINITE OR INFINITE=====
===Is an infinite Aquifer===
===Is a finite Aquifer===
===Is a finite Aquifer===
===Is a finite Aquifer===
===Is a finite Aquifer===
===Is a finite Aquifer===
===Is a finite Aquifer===
===Is a finite Aquifer===
===Is a finite Aquifer===
===Is a finite Aquifer===
===Is a finite Aquifer===
The dimensionless time values are:,
[0.0, 5.66862249281909, 11.33724498563818, 17.00586747845727, 22.67448997127636,
28.343112464095448, 34.01173495691454, 39.68035744973363, 45.34897994255272,
51.017602435371806, 56.686224928190896]

==============END================
The crossover times are:,
[0.19877879999999998, 0.6098529334495365, 1.239718427487128, 2.0919322677795527,
3.1690142581459124, 4.472927267793876, 6.0052832825402325, 7.767451993800409,
9.760625355004983, 11.985859190630363, 14.444101628247461, 17.13621340834513,
```

```
20.06298290822148, 23.225137574375108, 26.623352827591447, 30.258259137021614,
34.13044773467111, 38.240475298846135, 42.58886784130767, 47.17612396953358,
52.00271765163416, 57.06910058044037, 62.37570421090754, 67.92294152855882]

==============END=================
==The results shows a finite aquifer===
```

[7]:
```
# Claculating aquifer constant
def Aquifer_constant(
f = float(input('Enter f')),
phi = float(input('Enter phi')),
h = float(input("Enter h")),
Cw = float(input("Enter Cw")),
Cf = float(input("Enter Cf")),
ro = float(input("Enter ro"))):
    """
    f = o.389
    """

    Cfw = Cw + Cf
    U = 1.119*f*phi*h*Cfw* pow(ro, 2)
    print()
    print(f"The Aquifer constant value is: {round(U, 3)}")
    return U
U = Aquifer_constant()
U
```

```
Enter f0.389
Enter phi0.25
Enter h100
Enter Cw0.000003
Enter Cf0.000004
Enter ro9200

The Aquifer constant value is: 6447.53
```

[7]: 6447.530292000001

[8]:
```
# Calculating Dimensionless water influx, wD using rD = 10
def WrD_10(Aquifer_constant, W_D, t_D):
    We = U * change_in_pressure * W_D*(t_D)
    return We
```

5 Using Klins methods

[9]:
```
from math import *
from cmath import *


class Klin_et_al:
    # , rD, k, t, phi, Cw ,Cf, Cfw = Cw+Cf, ro, error
    def __init__(self,
                 rD=float(input("Enter the value of dimensionless radius,rD")),
                 k=float(input("Enter the value of permeability,k")),
                 t=int(input("Enter value of time, t")),
                 phi=float(input("Enter the value of phase angle, phi")),
                 Cw=float(input("Enter the value of water compressibility,␣
 ↪Cw")),
                 Cf=float(input("Enter the value of pore compressibility, Cf")),
                 ro=float(input("Enter the value of Oil radius, ro")),
                 f=float(input("Enter the value of f")),
                 h=float(input("Enter the value of the pay thickness")),
                 change_in_pressure=float(input("Enter the value of change in␣
 ↪pressure"))):
        """
        Using rD = 10 as the value of our rD
        k = permeability
        t = range of time
        phi = phase angle
        Uw = water viscosity
        Cfw = Total compressibility
        ro = Reservoir radius
        error = 0.0029%
        """
        self.rD = rD
        self.k = k
        self.t = t
        self.phi = phi
        self.Cw = Cw
        self.Cf = Cf
        self.Cfw = self.Cw + self.Cf
        self.ro = ro
        self.f = f
        self.h = h
        self.change_in_pressure = change_in_pressure

    #         self.error = 0.000029

    def tD(self):
        # for time in range(self.t):
```

```python
        tD = 2.309 * self.k * self.t / self.phi * self.Cw * self.Cfw * self.ro␣
↪** 2
        # print("At time(t):", time, "\t", "The dimensionless time is:",␣
↪round(tD, 2))
        return tD

    def t_cross(self):
        bo = -1.767
        b1 = -0.606
        b2 = 0.12368
        b3 = 3.02
        b4 = 2.25
        b5 = 0.50
        t_cross = bo + b1 * self.rD + b2 * self.rD ** b4 + b3 * (log(self.rD,␣
↪e)) ** b5
        if self.tD > self.t_cross:
            print("======The Aquifer is Finite Aquifer======")
        else:
            print("=======The Aquifer is infinite Aquifer======")
        return t_cross

    # Using the Appendix B approach to determine the qD
    def Apha_1(self):
        bo = -0.00222107
        b1 = -0.627638
        b2 = 6.277915
        b3 = -2.734405
        b4 = 1.2708
        b5 = -1.100417
        alpha_1 = bo + b1 * (1 / sinh(self.rD)) + b2 * self.rD ** b3 + b4 *␣
↪self.rD ** b5
        return alpha_1

    def Apha_2(self):
        bo = -0.00796608
        b1 = -1.85408
        b2 = 18.71169
        b3 = -2.758326
        b4 = 4.829162
        b5 = -1.009021
        alpha_2 = bo + b1 * (1 / sinh(self.rD)) + b2 * self.rD ** b3 + b4 *␣
↪self.rD ** b5

        return alpha_2

    # finding Jo amd J1
```

```
    """
    Jo = Bessel function of first kind order 0
    J1 = Bessel function of first kind order 1

    Since our aquifer is finite, it will be advisable to find Jo and J1 using
    x in range 0<= x <3. So, our x = 2

    Jo(x)  = bo +b1(x/3)**2 +b2(x/3)**4 +b3(x/3)**6 + b4(x/3)**8 +b5(x/3)**10 +b6(x/3)**12

    (x) -I*J1(x) = bo +b1 (x/3)**2 +b2(x/3)**4 + b3(x/3**6 + b4(x/3)**8 +b5(x/3)**10 + b6(x/3)**12,
    """

    def J_o(self):
        x = 2
        bo = 1.00
        b1 = -2.24999997
        b2 = 1.2656208
        b3 = -0.3163866
        b4 = 0.0444479
        b5 = -0.0039444
        b6 = 0.0002100

        Jo = bo + b1 * (x / 3) ** 2 + b2 * (x / 3) ** 4 + b3 * (x / 3) ** 6 + b4 * (x / 3) ** 8 + b5 * (x / 3) ** 10 + b6 * (x / 3)
        return Jo

    def J_1(self):
        x = 2
        bo = 0.5000
        b1 = -0.56249985
        b2 = 0.21093573
        b3 = 0.03954289
        b4 = 0.00443319
        b5 = -0.00031761
        b6 = 0.00001109

        J1 = (bo + b1 * (x / 3) ** 2 + b2 * (x / 4) ** 4 + b3 * (pow(x / 3, 6)) + b4 * (x / 3) ** 8 + b5 * (x / 3) ** 10 + b6 * (x / 3) ** 12)*x

        return J1

    # Finding the dimensionless flow rate

    def dimensionless_rate(self):
        Alpha_1 = self.Apha_1()
```

```
        Alpha_2 = self.Apha_2()
        tD = self.tD()
        J_1 = self.J_1()
        J_o = self.J_o()
        error = 0.000029
        qD_num1 = 2 * e ** (-Alpha_1 ** 2 * tD * J_1 ** 2 * (Alpha_1 * self.rD))
        qD_deno1 = (Alpha_1 ** 2 * (J_o ** 2 * Alpha_1 - J_1 ** 2 * (Alpha_1 *␣
↪self.rD)))

        qD_num2 = 2 * e ** (-Alpha_2 ** 2 * tD * J_1 ** 2 * (Alpha_2 * self.rD))
        qD_deno2 = Alpha_2 * (J_o ** 2 * Alpha_2 - J_1 ** 2 * (Alpha_2 * self.
↪rD))

        qD = self.rD ** 2 - 1 / 2 - qD_num1 / qD_deno1 - qD_num2 / qD_deno2 +␣
↪error
        return qD

    # Calculating aquifer constant
    def Aquifer_constant(self):
        B = 1.119 * self.f * self.phi * self.h * self.Cfw * pow(self.ro, 2)
        return B

    # Calculating Cumulative water influx using klin et al
    def We_10(self):
        B = self.Aquifer_constant()
        qD = self.dimensionless_rate()
        t_D = self.tD()
        We = B * self.change_in_pressure * qD * t_D
        return We


method = Klin_et_al()
print(method.We_10())
# rD = 10,k = 200, t = [* range (11)], phi = 0.25, Cw = 0.000003, Cf = 0.
↪000004, ro = 9200, error = 0.000029
```

```
Enter the value of dimensionless radius,rD10
Enter the value of permeability,k200
Enter value of time, t10
Enter the value of phase angle, phi0.25
Enter the value of water compressibility, Cw0.000003
Enter the value of pore compressibility, Cf0.000004
Enter the value of Oil radius, ro9200
Enter the value of f0.389
Enter the value of the pay thickness100
Enter the value of change in pressure225
(27231725768.127693+0j)
```

[10]:
```python
# The time and corresponding We using the Klin et al
t1 = 2995179062
t2 = 5158645218
t3 = 6639422824
t4 = 7522915487
t5 = 7839724010
t6 = 7947626041
t7 = 7341045140
t8 = 6326226015
t9 = 5172722501
t10 = 0
We_klins = {"We_klins": [2995179062,5158645218, 6639422824, 7522915487,␣
↪7839724010, 7947626041,
                        7341045140, 6326226015, 5172722501]}
df_we = pd.DataFrame.from_dict(We_klins)
pvt_data["Calculated_We"] = df_we["We_klins"]
pvt_data.head()
```

[10]:
```
   Time(yrs)  Pressure at O.W.C(psia)  Plateau Pressure Level(psia)  \
0          0                     2740                           NaN
1          1                     2500                        2620.0
2          2                     2290                        2395.0
3          3                     2109                        2199.0
4          4                     1949                        2029.0

   Change in Pressure(psia)  Np(MM/stb)  Rp(scf/stb)  Bo(rb/stb)  Rs(scf/stb)  \
0                     120.0         NaN          650       1.404          650
1                     225.0        7.88          760       1.374          592
2                     196.0       18.42          845       1.349          545
3                     170.0       29.15          920       1.329          507
4                     146.0       40.69          975       1.316          471

   Bg(rb/scf)      We  Calculated_We
0     0.00093     NaN   2.995179e+09
1     0.00098   3.829   5.158645e+09
2     0.00107  13.460   6.639423e+09
3     0.00117  26.462   7.522915e+09
4     0.00128  41.935   7.839724e+09
```

6 Performing history matching using graphical representaions

[16]:
```python
#fig, ax = plt.subplots(nrows = 2, ncols = 1, figsize = (10,6),␣
↪constrained_layout=True)
plt.figure(figsize = (10,7))
plt.plot(pvt_data["Change in Pressure(psia)"], pvt_data["Calculated_We"],␣
↪marker = "x", label = "Calculated Water influx");
```

```
plt.plot(pvt_data["Change in Pressure(psia)"], pvt_data["We"], marker = "+",␣
 ↪label = "Observed Water influx");
plt.xlabel("Change in pressure(psia)")
plt.ylabel("Water influx(bbl)")
plt.legend()
plt.show()
```