PETE 7242 Project

Daniel Barreca

Table of Contents

Executive Summary

Overview

This project involved the design of an integrated reservoir-wellbore-flowline model. The main task was to implement this model to study possibility of wax formation and to propose solutions for prevention of wax if needed. Flowrates of gas, liquid, and oil are also reported as a result of the flowline model.

Outline of Process and Development Stage

To gain understandings on what parts of the model may be improved and to verify which parts of the model are properly functioning, components of the code were implemented separately. These individual components were tested against various problems in literature. To further ensure validity, a grid independence study was conducted along with a comparison to an existing wellbore simulator. Finally, a sensitivity analysis was performed on various modifiable parameters to gain insight on what actions could be taken to prevent wax formation.

Statements about the Final Wellbore Model

While many assumptions have been made to simplify calculations in the integrated model, the simulator is still expected to give reasonably accurate results. However, improvements must be made before this wellbore simulator can be extended to other projects where different conditions exist. More details on what situations would not be applicable are listed within the report.

Summary of Validation and Production Rates

The grid independence study resulted in accuracy up to 1.7% relative error. For the base case, production rates were reported to be 1903 STB of oil per day, 275 barrels of water per day, and 1.99 MMSCF of gas per day. Production compared reasonably well with the benchmark simulator, reporting within 12.7% of the production and pressure values for various tested conditions.

Wax Prevention

Wax deposition is expected to form in a large section of the pipe. Changing riser diameter or wellbore insulation has minimal effect on temperature distribution within the pipe and riser. On the other hand, wax deposition can be entirely prevented by adding riser insulation or increasing flowrate through opening of the choke. From here, a decision can be made depending on additional information including:

- surface equipment limitations
- legal production quotas
- sand production expectations
- reservoir geology
- insulation material costs

Problem Statement

Background

Dolphin is a Gulf of Mexico crude oil field located 1935 feet below sea level and 18135 feet below seafloor level. Reservoir related details for calculation of IPR curve are listed in table 1.

Permeability	600 mD
Thickness	80 ft
Drainage Area	150 acres per well
Wellbore radius	6 inches
Drilling formation damage length	5 inches
Damaged formation permeability	100 mD
Reservoir temperature	225 ^{o}F
Reservoir Pressure	13,000 psia
Average Porosity	25%
Water Saturation	20%
Producing water cut	10%

Table 1: Reservoir Information

The flowline follows the seabed floor up to the subsea wellhead. From here, the wellbore dips nearly vertically (drilling through the seafloor) and continues its path down to the reservoir. The choke is located at the seafloor where the subsea wellhead is located. The well and flowline trajectory is listed in table 2.

Horizontal Location (ft)	Vertical Location (ft)
-3427.57	-20070
-1382	-11940
0	-1935
36115.2	-689
40180.8	-984
49209.6	-1050
63307.2	-820
65947.2	-394
66580.8	82
66950.4	82

Table 2: Well and Flowline Trajectory

Wellhead location is at a horizontal location of 0 and a vertical location of -1935 feet. Visualization of this pathway is displayed in Figure 1. Sea level is located at a vertical location of 0 ft.

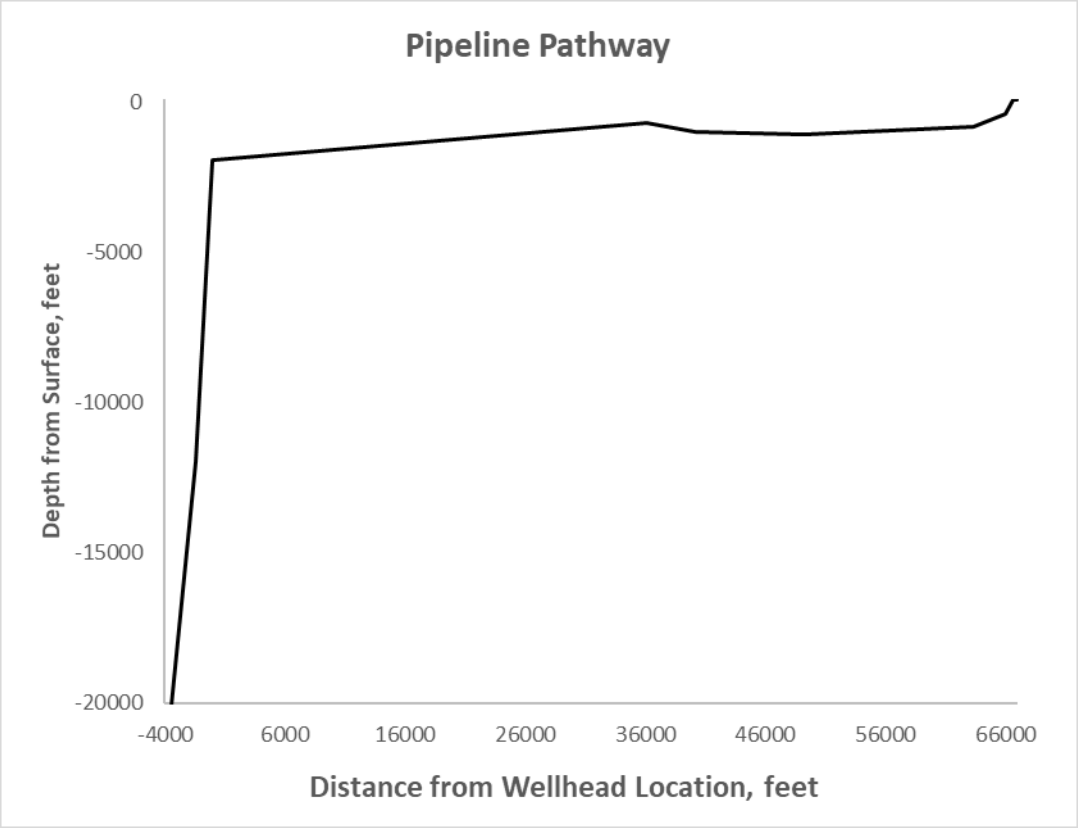

Figure 1: Pipeline Pathway

Basic PVT properties for use in PVT correlations are listed in Table 3.

Solution GOR	1050
Oil Gravity	35° API
Gas Gravity	0.7
Formation Water Specific Gravity	1.1
Wax Temperature Appearance (WAT)	110 °F
Solution GOR	1050
Oil Gravity	35° API

Table 3: Basic PVT Properties

In addition to PVT properties, reservoir information, and well trajectory, base case information is also provided at which flowrates are calculated.

Separator pressure	500 psi
Wellbore radius	6 inches
Choke opening	¼ inch
ID of flowline connecting subsea wellhead to host platform	5 inches
Types of completion	open hole

Table 4: Base Case

Other temperature and pipe geometry data are listed below:

Tubing ID, inches	4.4
Casing ID, inches	6
Sea Level Temperature, ^{o}F	80
Sea Floor Temperature, ^{o}F	43
Overall Heat Transfer Coefficent, Wellbore, $\mathbf{Btu/lbm - hr - ^{\circ} F}$	3
Overall Heat Transfer Coefficent, Flowline, $\mathbf{Btu/lbm - hr - ^{\circ} F}$	5

Table 5: Pipe Geometry and Temperature Related Information

Project Goals

Primary goals of this study include:

- Generate pressure and temperature profiles from reservoir to surface.
- Report production rates for oil, gas, and water.
- Analyze the possibility of wax formation and suggest actions for prevention if necessary.

Assumptions and Theory

Assumptions

Multiple assumptions were made to simplify the implementation of the integrated wellbore-reservoir simulation. Some of these assumptions are generally applicable to many situations. These assumptions include:

- Flow is assumed to be single phase for temperature calculation. Because flow is mainly single phase for this particular case, this assumption does not noticeably affect the results. Additionally, calculation of multiphase temperature is expected to be identical above a certain flowrate that this case meets.
- Analytic solution to temperature equation is used. Ideally, an option for rigorous enthalpy balance would be desired
- General fluid property correlations are used, such as Vasquez and Beggs [1]. that this reservoir is located in the Gulf of Mexico, it would have been more appropriate to use regional dependent correlations such as Petrosky [2].

Other assumptions are not sufficient and must be fixed before the simulator is used to analyze other cases:

- Relative permeability data is not included in the calculation of flowrate from the IPR model. This could significantly affect the predicted flowrate. Since the focus of this project in on the implementation of a wellbore simulation, this is temporarily ignored.
- Choke flow is assumed for the simulator. The user must take care here and double check the pressure profiles to ensure that flow is choked from the generated pressure profile.
- Assumes multiphase flow at choke. Additional equations should be included for calculation of single phase choking.
- Uses Hagedorn and Brown [3] for the horizontal section of calculation. This will introduce error as explained later.
- The simulator assumes ideal gas. Since the flowline is mostly single phase, this assumption should not greatly affect the results. Regardless, increased error will be introduced and this should be fixed before further use of the simulator.

Pressure Drop Calculation

Calculation of total pressure drop is broken up into three components (friction, gravity, and acceleration).

$$\frac{dP}{dz}\Big)_{total} = \frac{dP}{dz}\Big)_{friction} + \frac{dP}{dz}\Big)_{gravity} + \frac{dP}{dz}\Big)_{acceleration}$$

$$\frac{dP}{dz} = \frac{f_{TP}\rho_{TP}v_M^2}{2d} + \rho_{Slip}gsin\theta + \frac{\frac{\rho_{Slip}}{2}d(v_M^2)}{dz}$$

Effects of acceleration are neglected because pipe diameter is fairly constant and gas expansion is assumed to be negligible for this case (mostly liquid exists within the pipe). The equation is reduced to:

$$\frac{dP}{dz} = \frac{f_{TP}\rho_{TP}v_M^2}{2d} + \rho_{Slip}gsin\theta$$

For calculation of pressure drop within the wellbore and riser, the Hagedorn and Brown method is used. Hagedorn and Brown is a black box model developed from experimental data. Because of this, its application is limited to conditions similar to those experiments conducted. The Hagedorn and Brown method is generally applicable to vertical wells, fluid viscosities of 10-110 centipoise, API of 25-40, GLR below 5000, and tubing sizes of 1-1.5 inches in diameter. Calculation of components for the pressure gradients through Hagedorn and Brown is performed as follows:

$$\frac{dP}{dz} = \frac{f_{TP}\rho_{TP}v_M^2}{2d} + \rho_{Slip}gsin\theta$$

$$\rho_{TP} = \rho_{NS}^2/\rho_{Slip}$$

$$Re_{TP} = 1488\,\rho_{NS}v_M d/\mu_S$$

$$\mu_S = \mu_L^{H_L}\mu_G^{1-H_L}$$

$$CN_L = \frac{0.0019 + 0.0322N_L - 0.6642N_L^2 + 4.9941N_L^3}{1 - 10.0147N_L + 33.8696N_L^2 + 277.2817N_L^3}$$

$$H = \frac{N_{vl}P^{0.1}(CN_L)}{N_{vg}^{0.575}14.7^{0.1}N_D}$$

$$H_L = \Phi\left(\frac{0.0047 + 1123.32H + 729489H^2}{1 + 1097H + 722154H^2}\right)^{0.5}$$

$$B = \frac{N_{vg}N_L^{0.380}}{N_D^{2.14}}$$

$$\Phi = \frac{1.0896 - 69.9B + 2334B^2 - 12896B^3}{1 - 53.4B + 1518B^2 - 8420B^3}$$

Fluid Properties Calculation

Before calculating the pressure drop, fluid properties such as density and viscosity must be calculated. For this simulation, Vasquez and Beggs correlations are used. Vasquez and Beggs correlations are based on a wide variety of data, allowing them to be used generally rather than for specific regions. Bubble point pressure, solution gas oil ratio, and formation volume factor can be approximated through the following equations:

$$P_{bp} = \left(\frac{R_s}{C_1 * \gamma_g * e^{C_3\left(\frac{\gamma_o}{T+460}\right)}}\right)^{1/C_2}$$

Coefficient (Rs)	$\gamma_o < 30$	$\gamma_o > 30$
C_1	0.0362	0.0178
C_2	1.0937	1.1870
C_3	25.7240	23.9310

Table 6: Coefficients for calculation of solution gas oil ratio

$$R_s = C_1\gamma_g P^{C_2} e^{C_3\left(\frac{\gamma_o}{T+460}\right)}$$

When saturated, B_o can be calculated through

$$B_o = 1 + C_1R_s + C_2(T - 60)\left(\frac{\gamma_o}{\gamma_g}\right) + C_3R_s(T - 60)\left(\frac{\gamma_o}{\gamma_g}\right)$$

Coefficient (Bo)	$\gamma_o < 30$	$\gamma_o > 30$
C_1	$4.677 * 10^{-4}$	$4.670 * 10^{-4}$
C_2	$1.751 * 10^{-5}$	$1.100 * 10^{-5}$
C_3	$-1.811 * 10^{-8}$	$1.377 * 10^{-9}$

Table 7: Coefficients for calculation of formation volume factor

For undersaturated reservoirs:

$$B_o = B_{ob} e^{c_o(P_{bp} - P)}$$

Viscosity and correlations come from several sources and are described in Petroleum Production Systems. Gas is assumed to be ideal, therefore density is calculated assuming a compressibility factor of 1.

IPR Curve Calculation

Calculation of the Inflow Performance Relationship (IPR) curve is calculated through Vogel's Equation [5].

$$qL = J(P - P_b) + \frac{JP_b}{1.8}\left(1 - 0.2\left(\frac{P_{wf}}{P_b}\right) - 0.8\left(\frac{P_{wf}}{P_b}\right)^2\right)$$

For Vogel's equation, multiple assumptions are made:

- Circular bounded reservoir with a centered well
- The porous medium is uniform and isotropic
- Compressibility and gravity is neglected
- Capillary pressure is neglected

Choked Flow Calculations

Choked and multiphase flow is assumed regardless of actual conditions. For this calculation, Gilbert's equation [6] is utilized:

$$P_{upstream} = A q_L (GLR)^B / D_{64}^C$$

Where coefficients:

Coefficient	
$\boldsymbol{A}$	10
$\boldsymbol{B}$	0.546
$\boldsymbol{C}$	1.89

Table 8: Coefficients for calculation of Gilbert's Equation

Though choke flow is assumed to always be present, this equation is only valid when $P_{downstream}/P_{upstream} < 0.7$

Temperature Curve Calculation:

For estimation of temperature distribution within the pipe, an analytical approximation from Brill and Mukherjee [7] is used. Flowing fluid is assumed to be single phase. Temperature within the pipe is dependent on multiple parameters, most notably the mass flow rate, overall heat transfer coefficient, and geothermal gradient of the surrounding environment.

$$A = \frac{C_p w}{U \pi d}$$

$$T_f = (T_{ei} - g_G L sin\theta) + (T_i - T_{ei}) e^{-\frac{L}{A}} + g_G sin\theta A \left(1 - e^{-\frac{L}{A}}\right)$$

For multiphase flow, other expressions explained by Hasan and Kabir [8] may be used. Because mass flow rate of liquid is magnitudes larger than that of gas for this particular case, these multiphase expressions are not used and single phase is assumed.

Algorithm and Implementation

Equations are implemented using the algorithm below.

Figure2: Algorithm for Calculation of Flowing Well Pressure

Results

Validation

A grid independence study is conducted to ensure that grid resolution adequately resolved. Both temperature and pressure graphs are generated to analyze this. Figures 3-5 in Appendix “A” demonstrate the results. After 300 nodes, solution begins to converge to a sufficient level. It can be seen in Figure 4 that the temperature difference between 300 nodes and 1000 nodes is only 1°F, or 1.7% relative error. Similar results repeat themselves in Figure 5 for the pressure distributions.

Pressure distribution and liquid flowrate is also compared to another pipeline simulator.

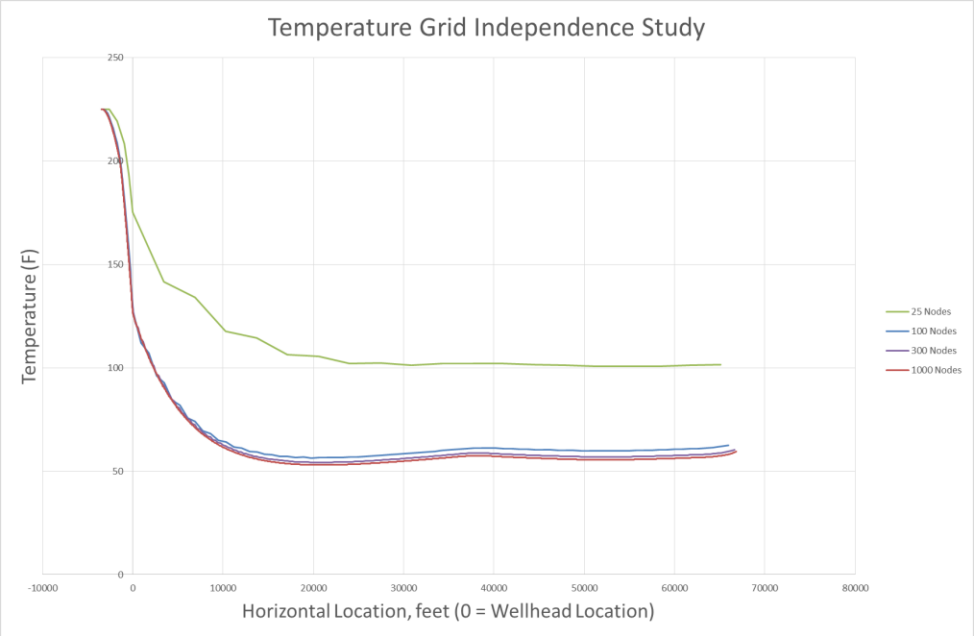

Figure 3: Temperature Grid Independence Study

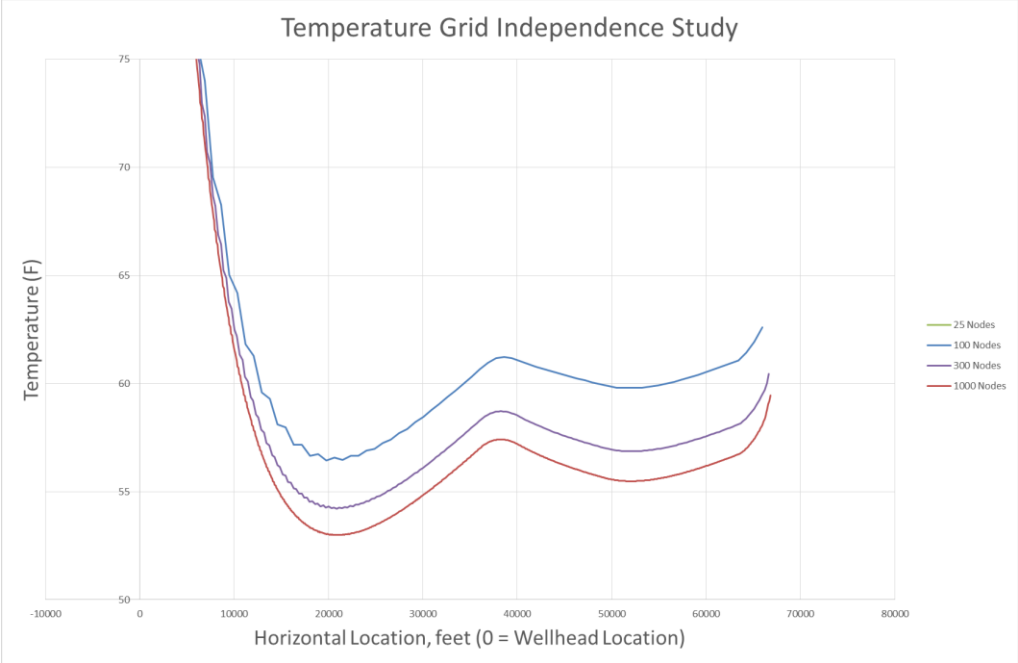

Figure 4: Temperature Grid Independence Study –Zoomed View

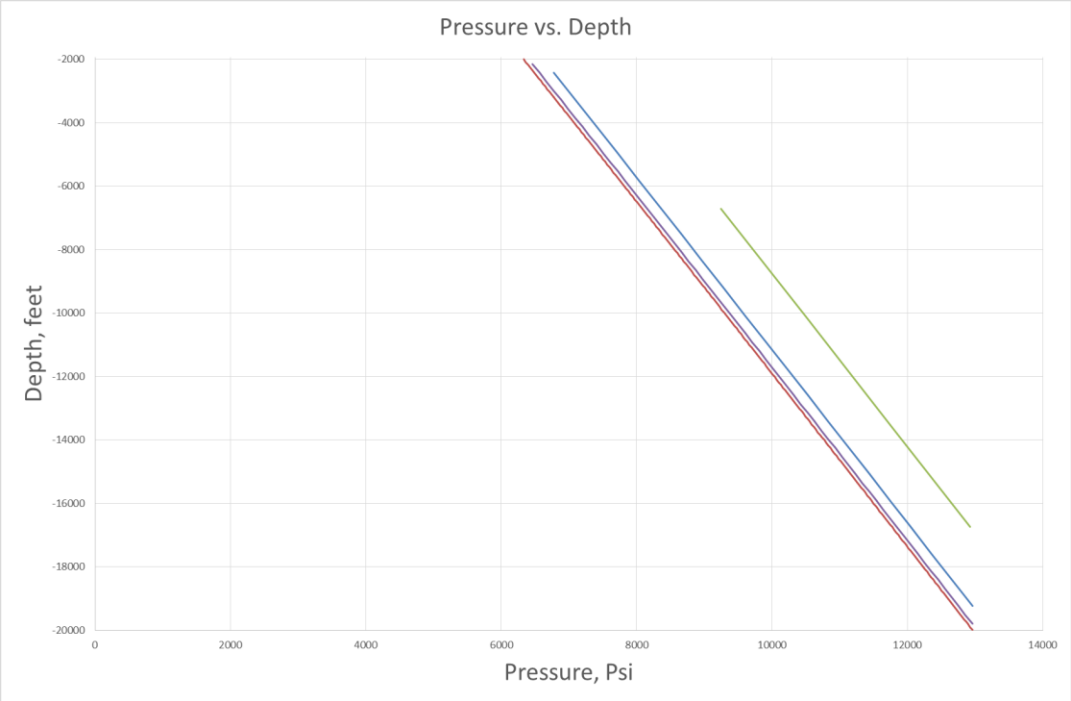

Figure 5: Pressure Grid Independence Study

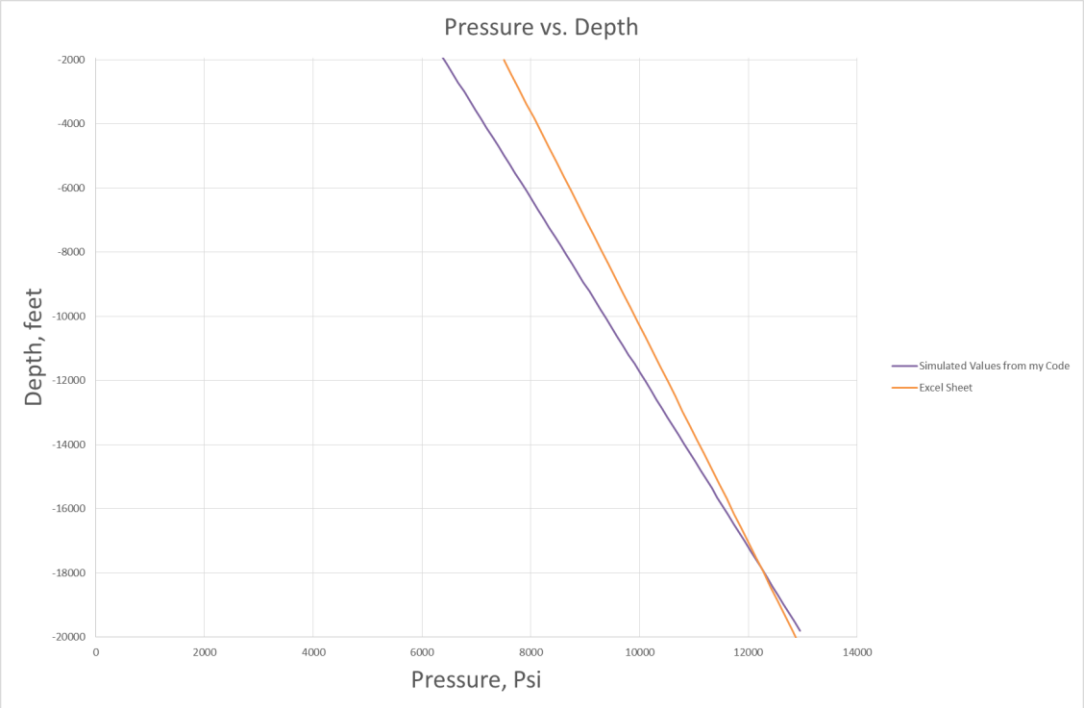

Figure 6: Validation Study- Comparison to benchmark simulator

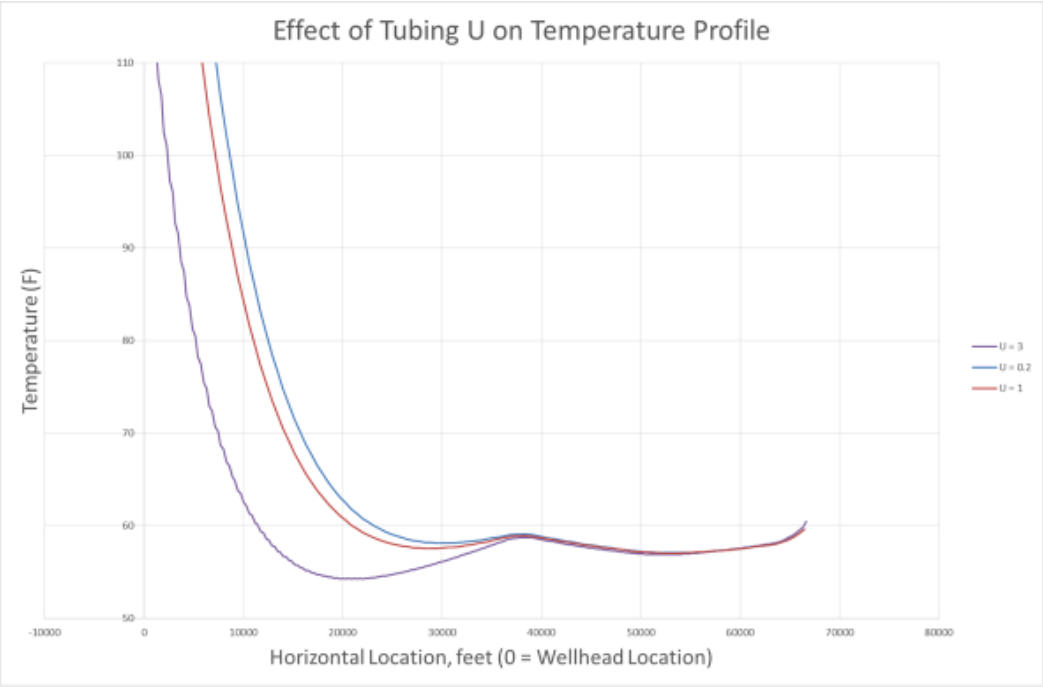

Figure 7: Effect of Wellbore Overall Heat Transfer Coefficient

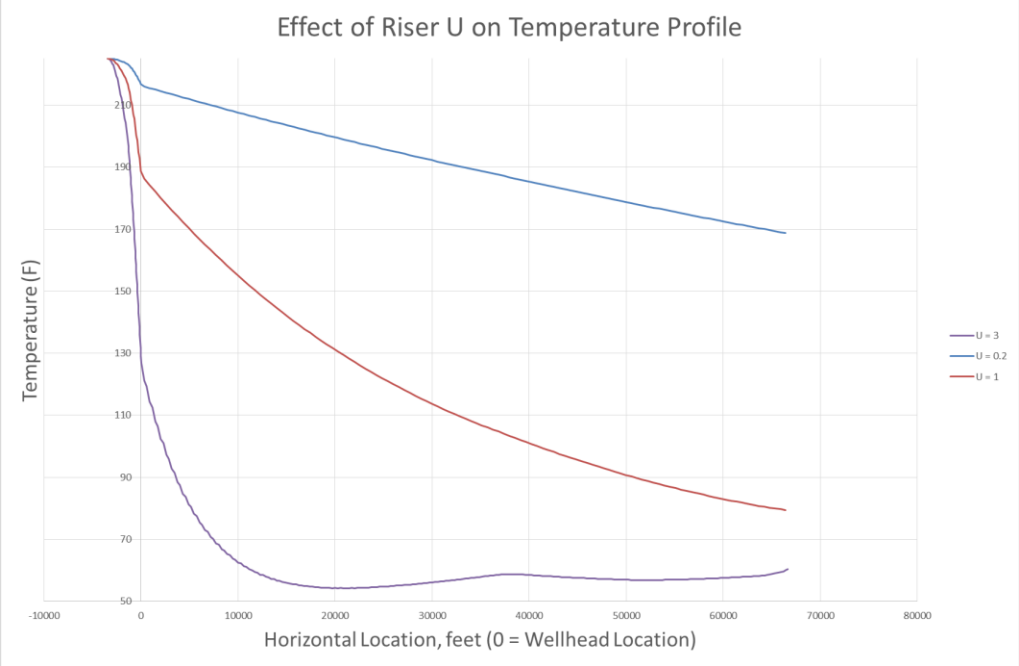

Figure 8: Effect of Flowline Overall Heat Transfer Coefficient

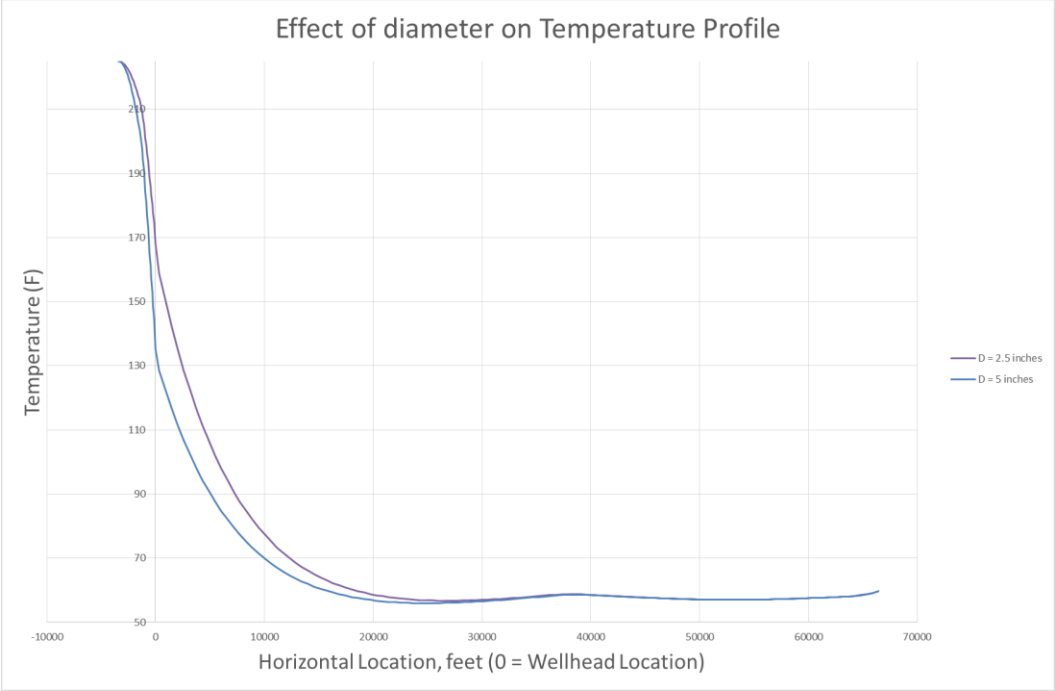

Figure 9: Effect of Flowline Diameter on temperature distribution

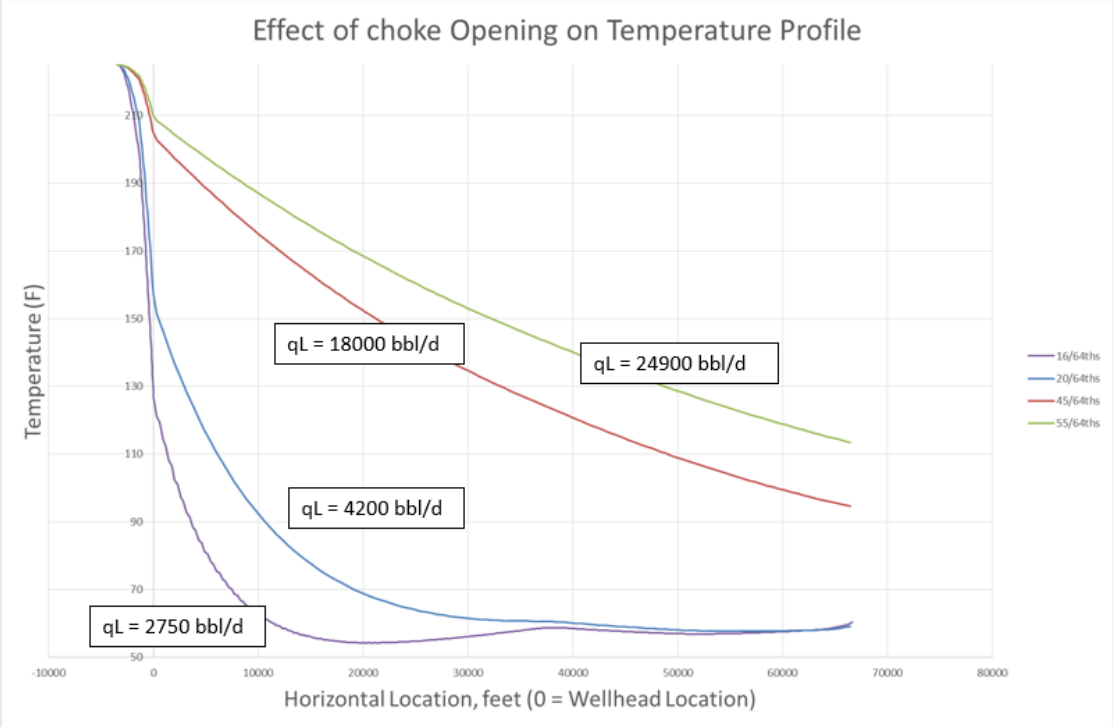

Figure 10: Effect of Choke Opening on Temperature Distribution

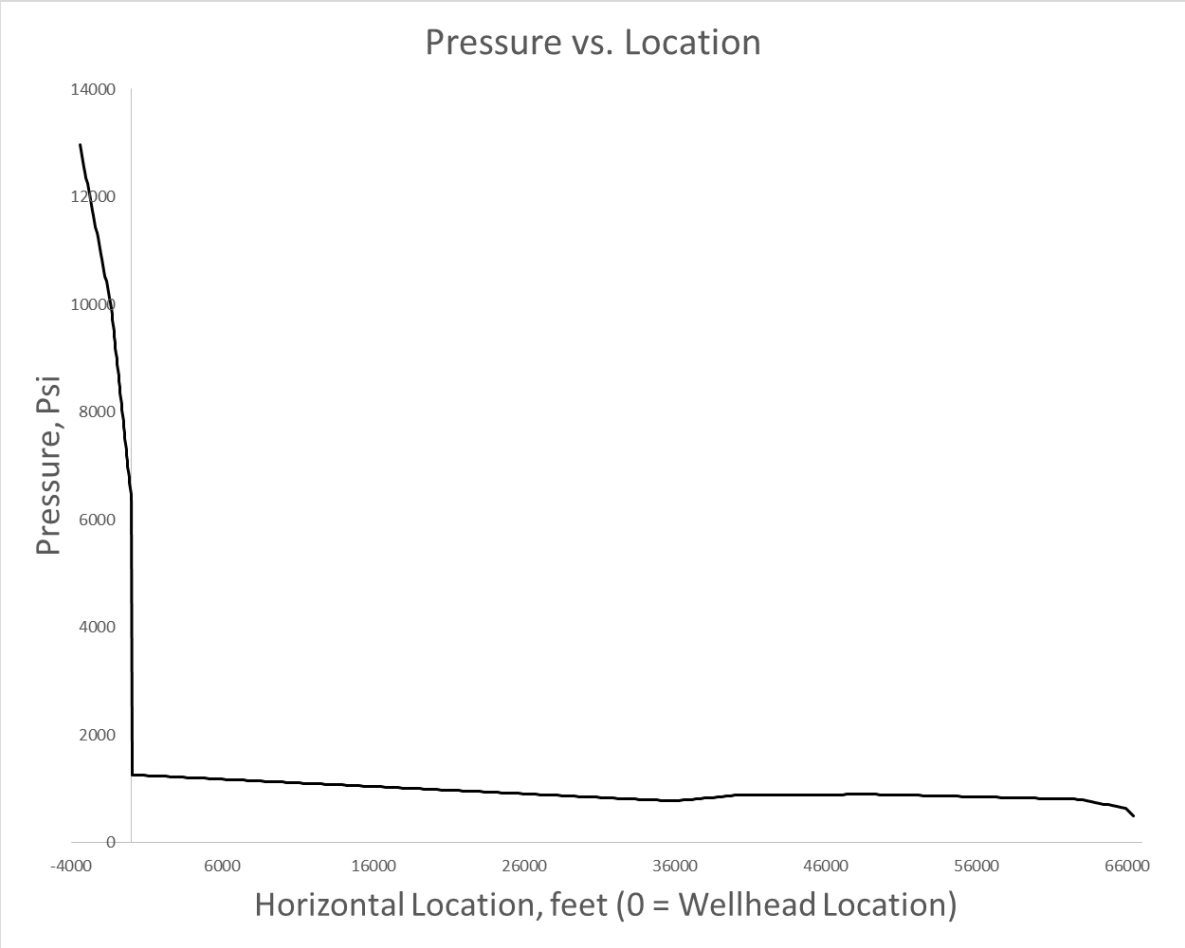

Figure 11: Pressure Distribution

$\boldsymbol{\mu} = viscosity, cp$

$\boldsymbol{\rho} = density, lbm/ft\text{^}3$

$\boldsymbol{B_o} = volume\ factor$

$\boldsymbol{C_P} = heat\ capacity\ of\ fluid, Btu/lbm - ^\circ F$

$\boldsymbol{d} = diameter, inches$

$\boldsymbol{g_G} = geothermal\ gradient, ^\circ F/ft$

$\boldsymbol{f} = friction\ factor$

$\boldsymbol{P} = Pressure, psi$

$\boldsymbol{q} = volume\ flowrate, bbl/d$

$\boldsymbol{Re} = Reynold's\ Number$

$\boldsymbol{T_{ei}} = Temperature\ of\ undisturbed\ surroundings, ^\circ F$

$\boldsymbol{T_i} = Fluid\ Temperature\ at\ previous\ location, ^\circ F$

$\boldsymbol{T_f} = Fluid\ Temperature\ at\ next\ location\ to\ be\ calculated, ^\circ F$

$\boldsymbol{U} = Overall\ Heat\ Transfer\ Coefficient, Btu/lbm - hr - ^\circ F$

$\boldsymbol{w} = Mass\ flowrate, lbm/hr$

References

[1] Vazquez, Milton, and H. Dale Beggs. "Correlations for fluid physical property prediction." SPE Annual Fall Technical Conference and Exhibition. Society of Petroleum Engineers, 1977.

[2] Petrosky Jr, G. E., and F. F. Farshad. "Pressure-volume-temperature correlations for Gulf of Mexico crude oils." SPE annual technical conference and exhibition. Society of Petroleum Engineers, 1993.

[3] Hagedorn, Alton R., and Kermit E. Brown. "Experimental study of pressure gradients occurring during continuous two-phase flow in small-diameter vertical conduits." Journal of Petroleum Technology 17.04 (1965): 475-484.

[4] Economides, Michael J., A. Daniel Hill, and Christine Ehlig-Economides. Petroleum production systems. Pearson Education, 2012.

[5] Vogel, J. V. "Inflow performance relationships for solution-gas drive wells." Journal of petroleum technology 20.01 (1968): 83-92.

[6] Gilbert, W. E. "Flowing and gas-lift well performance." Drilling and production practice. American Petroleum Institute, 1954.

[7] Brill, James P., and Hemant Kumar Mukherjee. Multiphase flow in wells. Vol. 17. Society of Petroleum Engineers, 1999.

[8] Hasan, A. Rashid, C. Shah Kabir, and Cem Sarica. Fluid flow and heat transfer in wellbores. Richardson, TX: Society of Petroleum Engineers, 2002.


```
#include <iostream>
#include <iomanip>
#include <fstream>
#include <vector>
```

```
#include <math.h>
#include <vector>
#include <stdlib.h>
using namespace std;
int main()
{



#include "Declarations.H"
#include "InputDeck.H"
#include "assignThetas.H"  /*Creates an array of theta values based on WellTrajectories*/


Pu = Pwf + 10000;
gammaO = 141.5 / (APIOil + 131.5);

while (Pu-PFinal>10){
        #include "IPR.H"
        #include "chokeEqn.H"
        double TestL = 0;

        /*--CLEAR----
        -----ALL-----
        ---VECTORS---
        -----HERE--*/
        Tf.clear();
        TubingP.clear();
        Tf.push_back(TRes);

        /*Calculate some initial fluid properties at reservoir Temp because
        we need it to calculate density for the TEqn*/
        T = TRes;
        P = PRes;
        #include "FluidProperties.H"

        T1 = TRes;


        for(int i = 0; i < WHL+1; i = i + 1 ) {
                if (i == 0) {
```

```
                TubingP.push_back(Pwf);
                Tf.push_back(TRes);
                TestL = TestL + dL;
        qLi = qL;
        rhoLi = rhoL;
          }
          else{
                POld = TubingP[i-1];
                PNext = POld - POld*dL / Length; /*For the guess, assume linear pressure
distribution*/
                Pguess = PNext + 50;
                counter = 0;
                Tei = T1;
                T1 = T1 - gG*dL*sinThetas[i];
        TEnv.push_back(T1);
                while (abs(Pguess - PNext) > 1 || counter > 10000){
                        Pguess = PNext;
                        /*Calculate the temperature based on analytical Solution*/
                        #include "TEqn.H"
                        /*Calculate Fluid Properties based on Temp and Pressure*/
                        P = TubingP[i-1];
                        T = Tf[i-1];
                        #include "FluidProperties.H" /*use POld**/
                        /*Calculate dPdL using H&G*/
                        #include "HagedornAndBrown.H"
                        PNext = TubingP[i-1] - dPdL * dL;
                        counter = counter+1;
                                cout << "counter ====================== " << counter
<< endl;
                        cout << "TubingP= " << TubingP[i] << endl;
                        cout << "PNext= " << PNext << endl;
                        cout << "POld= " << POld << endl << endl;
                        cout << "Pguess= " << Pguess << endl << endl;
                        cout << "Temp= " << TemporaryT << endl << endl;
                }

                TubingP.push_back(PNext);
                Tf.push_back(TemporaryT);
        rhoLvec.push_back(rhoL);

          }
```

```
        }
        PFinal = TubingP[WHL-2];
        Pwf= Pwf+1;
}



for(int i = WHL+1; i < N; i = i + 1 ) {

   if (T1 > 210){
      T1 = Tf[i-1];
   }
        Tei = T1;
        T1 = T1 - gG*dL*sinThetas[i];
      TEnv.push_back(T1);
        /*Calculate the temperature based on analytical Solution*/
        #include "TEqn.H"
        Tf.push_back(TemporaryT);
}

double PTemp = Psep;
vector<float> TTop;

for(int i = N; i > WHL+1; i = i - 1 ) {
      #include "HagedornAndBrown.H"
        PTemp = PTemp + dPdL * dL;
        TTop.push_back(PTemp);
}

for(int i = N-WHL-2; i > 0; i = i - 1 ) {
        TubingP.push_back(TTop[i]);
}


cout << "temp | Location = " << endl;
for(int i = 0; i < N; i = i + 2 ) {
        cout << Tf[i] << "   "  << Location[i] <<endl;
}
```

```
cout << "Pressure | dens | Location = " << endl;
for(int i = 0; i < N; i = i + 1 ) {
        cout << TubingP[i] << "   "  << verti[i] <<endl;
}



/*
   if pu / pd < 0.5

        petroleum production systems
        chapter 3
        3-27 correct


        typo in appendix B-

*/
cout << Pwf << " Final Result" << endl;
cout << PFinal << " WellHead pressure" << endl;

cout << qL << " Fluid rate" << endl;

cout << WHL << endl;
        getchar();
   return 0;
}


// ********************************************************************** //

// ********************************************************************** //
********************************************************************** //

ifstream inFile;
inFile.open("inputDeck.dat");
string vert;
string hor;

int vectorSize = 0;
```

```
if (!inFile)
	{
	cout << "Can't open or find inputDeck.dat";
	 // terminate with error
	}

	/*
	 * Reads a text file and outputs the text on a screen
	 */

	string word;
	string Search;

	int count = 0;

	if (inFile.is_open())
	{
		/*
		* Loop that goes to end of file
		*/
		while ( inFile.good() )
		{
			if (word == "WELL_LENGTH")
			{
			   inFile >> Length;
			}

			if (word == "NUMBER_OF_DIVISIONS")
			{
			   inFile >> N;
			}

			if (word == "WellTrajectory")
			{
				while (vert != "END")
				{
					inFile >> hor;
						if (hor == "END"){
							break;
						}
					inFile >> vert;
```

```
                    vertLoc.push_back(strtof((vert).c_str(),0));
                    horLoc.push_back(strtof((hor).c_str(),0));
                    vectorSize = vectorSize + 1;
                }
            }

if (word == "TubingID")
            {
               inFile >> TubingID;
            }

            if (word == "CasingID")
            {
               inFile >> CasingID;
            }

            if (word == "SeaBedTemp")
            {
               inFile >> TSeaBed;
            }

            if (word == "SeaLevelTemp")
            {
               inFile >> TSeaLevel;
            }

if (word == "USEA")
            {
               inFile >> USea;
            }

            if (word == "USURROUNDING")
            {
               inFile >> USurr;
            }

            if (word == "PERM")
            {
               inFile >> Perm;
            }
```

```
            if (word == "THICKNESS")
            {
               inFile >> thickness;
            }

if (word == "DRAINAGE")
            {
               inFile >> DrainageArea;
            }

            if (word == "RWB")
            {
               inFile >> rwb;
            }

            if (word == "DAMAGELENGTH")
            {
               inFile >> DLength;
            }

            if (word == "DAMAGEPERM")
            {
               inFile >> DPerm;
            }

if (word == "RESERVOIRTEMP")
            {
               inFile >> TRes;
            }

            if (word == "RESERVOIRPRESSURE")
            {
               inFile >> PRes;
            }

            if (word == "POROSITY")
            {
               inFile >> Porosity;
            }

            if (word == "WATERSATURATION")
```

```
            {
               inFile >> Sw;
            }

if (word == "PRODUCINGWATERCUT")
            {
               inFile >> WCut;
            }

            if (word == "PRODUCINGGOR")
            {
               inFile >> Rp;
            }

            if (word == "OILGRAVITY")
            {
               inFile >> APIOil;
            }

            if (word == "GASGRAVITY")
            {
               inFile >> gammaG;
            }

            if (word == "FORMATIONWATERGRAVITY")
            {
               inFile >> gammaW;
            }

            if (word == "WAXTEMP")
            {
               inFile >> TWax;
            }

            if (word == "SEPARATOR_TEMPERATUR")
            {
               inFile >> Tsep;
            }

            if (word == "SEPARATOR_PRESSURE")
            {
```

```
                    inFile >> Psep;
                }

            if (word == "WELLHEAD_LOCATION")
                {
                    inFile >> WH_Location;
                }

                if (word == "PWF_Guess")
                {
                    inFile >> Pwf;
                }

                if (word == "CHOKE_OPENING")
                {
                    inFile >> CHO;
                }

                inFile >> word;
            }
            inFile.close();
        }

  else cout << "Unable to open file";

        /*
        while (inFile >> shift)
        cout << shift << endl;
        */

        inFile.close();


/*--------------------TEQN-----------------------------------------------------*/
/*A = Cp * w / U / 3.14 / d*/



gGSea = -0.0121;
gGEarth = 0.01;
```

```
if (Location[i] > WH_Location){
        gG = gGSea;
        U = USea;
        d = TubingID/ 12;
}
else{
        gG = gGEarth;
        U = USurr;
        d = CasingID/12;
}

/*Lr = 2 * 3.14 *d/2 * U / Cp / w*/
w = rhoLi * qLi;

A = Cp * w / U / 3.14 / d*5.614/24;

cout << "w----------" << w << endl;
cout << "T1"<< T1 << endl;
cout << "Tei"<< Tei << endl;




TemporaryT = T1 + (Tf[i-1]-Tei)*pow(2.71828,(-dL/A)) + gG*sinThetas[i]*A*(1-pow(2.71828,(-
dL/A)));


cout << "Tei----------"<< Tf[i-1] << endl;



/*
rhoL = function of Temp and Pressure at the grid Block
rhoG = function of Temp and Pressure at the grid Block
muL = function of Temp and Pressure at the grid Block
muG = function of Temp and Pressure at the grid Block
Bg
Rs
#include FluidProperties.H
```

```
*/
Area = 3.14 / 4 * pow(d,2);
qG = (Rp - Rs) * Bg*qL*(1-WCut);
vsl = (qL) / Area / 86400*5.614;
vsg = qG / Area / 86400;

cout << qL << " ------------------------qL" << endl;
cout << qG << " ------------------------qG" << endl;

/*Prevents a nan error*/
if (vsg <= 0.001){
        vsg = 0.0001;
}



vm = vsg+vsl;
gammaL = vsl / vm;
muL = 2;
muG = 0.019;
sigma = 31.6;

;
double Pwh = 500;

NLV = 1.938 * vsl * pow((rhoL/sigma),0.25);
Nd = 120.872 * d * pow((rhoL/sigma),0.5);
NGV = 1.938 * vsg * pow((rhoL/sigma),0.25);
NL = 0.15726 * muL * pow((1/(rhoL*pow(sigma,3))),0.25);



CNL = 0.0019 + 0.0322*NL - 0.66422*pow(NL,2) + 4.9951 * pow(NL,3);
CNL = CNL / (1 - 10.0147*NL+33.8696*pow(NL,2)+277.2817*pow(NL,3));

/*What is P here*/
H= NLV * pow((Pwh / 14.7),0.1) * CNL / pow(NGV,0.575) / Nd;
B = NGV * pow(NL,0.380) / pow(Nd,2.14);

phi = 1.0886 - 69.9473*B + 2334.3497*pow(B,2) - 12896.683*pow(B,3);
phi = phi / (1 - 53.4401 * B + 1517.9369*pow(B,2) - 8419.811*pow(B,3));
```

```
HL = (0.0047 + 1123.32 * H + 729489.64*pow(H,2));
HL = HL / (1+1097.1566*H + 722153.97*pow(H,2));
HL = phi*pow(HL,0.5);


/*-------------------------IMPORTANT----------------------------------------------------*/
/*bug in my code??? HL can go above 1 in certain cases... I need to check this out later...
/*-------------------------------------------------------------------------------------*/
if (HL > 1){
HL = 1;
}



muSlip = pow(muL,HL) * pow(muG,(1-HL));
rhoSlip = rhoL*HL + rhoG * (1-HL);
rhoNS = rhoL*gammaL + rhoG * (1-gammaL);

ReTP = 1488 * rhoNS * vm * d / muSlip;
fTP = 0.184*pow(ReTP,-0.2);




rhoTP = pow(rhoNS,2) / rhoSlip;
dPdL = (fTP * rhoTP * pow(vm,2)) / (2 * g * d); /*friction component*/

dPdL = dPdL + rhoSlip * 1 * sinThetas[i]; /*Gravity Component*/

dPdL = dPdL / 144; /*Convert to psi/ft from psf/ft*/

cout << "HL iis = " <<  HL << endl;
cout << "HL iis = " <<  rhoO << endl;
cout << "HL iis = " <<  gammaO << endl;



/*Fluid Properties Formulas*/
```

```
/*Petrosky*/




/*----------------------------------------------------------------------------------------------------------
----------------------------------------------OIL-----------------------------------------------------------
----------------------------------------------PROPERTIES----------------------------------------------------
----------------------------------------------------------------------------------------------------------*/



/*----------------------------------------------------------------------------------------------------------
-------------------------------------------Calculate Solution GOR-------------------------------------------
----------------------------------------------------------------------------------------------------------*/

if (APIOil > 30){
     A = 1.0937;
     B = 27.64;
     C = 11.172;
}
else{
    A = 1.187;
    B = 56.06;
    C = 10.393;
}
gammaG100 = (1. + 0.5912 * APIOil * Tsep * 0.0001 * log10(Psep / 114.7) /
log10(10))*gammaG;
Rs = (gammaG100 * pow(P,A) / B);
Rs = Rs * pow(10,C*APIOil/(T+460));
Rs = 1050;
Pb = B*Rp;
Pb = Pb / gammaG100 / pow(10,(C * APIOil / (T+460)));
Pb = pow(Pb,(1/A));


if (Rs < 0){
Rs = 0;
}
```

```
if (Rs>Rp){
Rs = Rp;
}




/*-----------------------------------------------------------------------------------------------------------
----------------------------------------------Bubble Point Pressure-----------------------------------------
-----------------------------------------------------------------------------------------------------------*/
if (APIOil <= 30){
        C1 = 0.0362;
        C2 = 1.0937;
        C3 = 25.7240;
}
else{
        C1 = .0178;
        C2 = 1.1870;
        C3 = 23.9310;
}
/*
Pb = pow((Rp / C1 / gammaG / pow(2.71828,C3*APIOil/(T+460))),1/C2);
*/


/*-----------------------------------------------------------------------------------------------------------
----------------------------------------------Calculate Oil Viscosity---------------------------------------
-----------------------------------------------------------------------------------------------------------*/
/*Beggs and Robinson*/
muOd = 31410000000 * pow(T , (-3.444)) * pow((log(APIOil) / log(10)) , (10.313 * log(T) /
log(10) - 36.447));
A = 10.7150 / pow((Rs + 100) , 0.515);
B = 5.440 / pow((Rs + 150) , 0.338);
muO = A * pow(muOd , B);

if (P > Pb) {
    A = 2.6 * pow(P , 1.187) * pow(10 , (-0.039 * P * 0.001 - 5));
    muO = muO * pow((P / Pb) , A);
}
```

```
/*-----------------------------------------------------------------------------------------------------------
----------------------------------------------Calculate Liquid Density----------------------------------------
-----------------------------------------------------------------------------------------------------------*/
/*Vasquez*/
gammaO = 141.5 / (131.5 + APIOil);
D = (T - 60) * APIOil / gammaG100;
if (APIOil <= 30) {
   A = 0.1751;
   B = -1.8106;
}
else{
   A = 0.11;
   B = 0.1337;
}

if (P < Pb){
        Bo = 1 + 0.000467 * Rs + A * D * 0.0001 + B * Rs * D * 0.00000001;
   CO = (-1433 + 5 * GOR + 17.2 * T - 1180 * gammaG100 + 12.61 * APIOil) / (P * 100000);
}
else{
        Bob = 1 + 0.000467 * GOR + A * D * 0.0001 + B * GOR * D * 0.00000001;
        CO = (-1433 + 5 * Rs + 17.2 * T - 1180 * gammaG100 + 12.61 * APIOil) / (P * 100000);
        Bo = Bob * pow(2.718,(CO * (Pb - P)));
}



/*gammaG???*/
rhoO = (gammaO * 62.4) / Bo;
rhoW = gammaW * 62.4;
rhoL = rhoO * (1-WCut) + rhoW * WCut;




/*-----------------------------------------------------------------------------------------------------------
-----------------------------------------------------GAS-------------------------------------------------
```

```
----------PROPERTIES----------
----------*/

/*----------
----------Calculate Bg----------
----------*/
Z = 1; /*assume ideal gas for now*/
Bg = 0.0283 * Z * (T+460) / P;

/*----------
----------Calculate Gas Density----------
----------*/
rhoG = P/10.7315/(T+460)/ (28.9586*gammaG); /*lbm/ft^3*/

/*----------
----------Calculate Gas Viscosity----------
----------*/
Ma = 28.9586*gammaG;
A = (9.379 + 0.01607 * Ma) * pow(T+460,1.5);
A = A / (209.2 + 19.26*Ma+T+460);
B = 3.448 + 986.4 / (T+460) + 0.01009 * Ma;
C = 2.447 - 0.2224 * B;
muG = A * 0.0001 * exp(B*pow(rhoG*0.0160,C));

/**********
**********
*/
cout << gammaG100 << " ----------gamma" << endl;
cout << Pb << " ----------Pb" << endl;
cout << Rs << " ----------Rs" << endl;
cout << muO << " ----------muO" << endl;
cout << rhoL << " ----------rhoL" << endl;
cout << Bo << " ----------Bo" << endl;

/*----------Input Deck----------*/
double Length;                  /*feet*/
```

```
int N;                  /**/
double dL;              /*feet*/
vector<float> vertLoc;          /*feet*/
vector<float> horLoc; /*feet*/
double TubingID;        /*inch*/
double CasingID;        /*inch*/
double TSeaBed;                 /* F */
double TSeaLevel;       /* F */
double USea;            /**/
double USurr;           /**/
double Perm;            /* mD */
double thickness;       /*feet*/
double DrainageArea;/*feet^2*/
double rwb;             /*inch*/
double DLength;                 /*inch*/
double DPerm;                   /* mD */
double TRes;            /* F */
double PRes;            /* psia */
double Porosity;        /**/
double Sw;              /**/
double WCut;            /**/
double GOR;             /**/
double APIOil;          /* API */
double gammaG;                  /**/
double gammaW;                  /**/
double TWax;            /* F */
double Tsep;
double Psep;

/*---------------------------------END INPUT DECK-----------------------------------------------------------------*/



double A; /*coefficient for choke equation*/
double B; /*coefficient for choke equation*/

double Bo; /*oil formation volume factor as function of pressure and temp*/
double Bg; /*gas formation volume factor as function of pressure and temp*/

double C; /*coefficient for choke equation*/
double CNL; /*Coefficient for H&B*/
```

```
double d; /*tubing diameter*/
double dPdL; /* pressure drop*/
double D; /*choke diameter in 64ths of an inch*/

double fTP; /*friction factor for use in hagedorn and brown*/
double gammaL; /*no slip liquid holdup*/
double g = 32.2;
double GLR;

double HL; /*Liquid holdup*/

double muL; /*viscosities in cp*/
double muG;
double muO;
double muSlip;

/*Dimensionles numbers for hagedorn and Browm */
double Nd;
double NGV;
double NL;
double NLV;

double phi; /*Coefficient for Hagedorn and Brown*/
double P; /*What is P here???*/
double Pu; /*pressure upstream from choke */
double PNext;
double POld;
double Pguess;
double qL; /* liquid flowrate*/
double qO;
double qG;

double ReTP; /*Reynolds number for use in hagedorn and brown*/

double rhoL; /*liquid density in lbm/ft^3*/
double rhoLBH; /*liquid density at bottomhole*/
double rhoG;
double rhoTP;
double rhoNS;
double rhoSlip;
```

```
double Rs; /*Solution gas oil ratio as function of temp and Pressure*/
double sigma; /*surface tension in dynes*/

double theta; /*inclination angle*/
vector<float> TubingP;

double vslBH; /*superficial Velocity at bottomhole*/
double vsl; /*superficial velocities*/
double vsg;
double vm;



/*-----------------FLUID PROPERTIES-------------------------*/
double Bob;
double Co;      /*undersaturated oil compressibility*/
double gammaO;  /*Oil gravity*/
double Pb;      /*Bubble Point Pressue*/
double Rsb;     /*Solution Gas Oil ratio at bubble point*/
double X;       /*Random parameter*/

double a;
double C1;
double C2;
double C3;
double CO; /*oil compressibility*/
double gammaG100;
double gammagd;
double Rp;
double rhoO;
double rhoW;
double muOb;
double muOd;
double b;
double m;
double Z;
double T;
double Ma;
```

```
/*Temp Equation*/
vector<float> Tf;
double T1;
double Tei;
double TemporaryT;
double gGEarth;
double gGSea;
double U;
double w;

double gG;
double Cp = 0.65;
/*
gGEarth = ...
dGSea = ...*/



/**/
double WH_Location;
double BHagedorn;
double H;
double Area;
double PFinal;
double counter;
double J;
double WHL;
double Pwf;
double CHO;
double rhoLi;
double qLi;
vector<float> rhoLvec;
vector<float> TEnv;


/*Gilbert*/
A = 10;
B = 0.546;
C = 1.89;
```

```
Pu = A * qL * pow(Rp,B) / pow(CHO,C);

cout << "Pu here ----------" <<  Pu << endl;

double rise;
double run;
double hypotenuse;
double height;
double distance;
vector<float> sinThetas;
vector<float> cosThetas;
vector<float> Location;
vector<float> verti;
double L;
L = 0;
Length = 0;
distance = horLoc[0];
Location.push_back(horLoc[0]);
height = vertLoc[0];
int j;

for(j = 0; j < vectorSize-1; j = j + 1 ){
        rise = (vertLoc[j]-vertLoc[j+1]);
    run = (horLoc[j] - horLoc[j+1]);
        hypotenuse = pow((pow(rise,2)+pow(run,2)),0.5);
        Length = Length + hypotenuse;
}

dL = Length/(N);


height = vertLoc[0];
distance = horLoc[0];

j = 0;
for(int i = 0; i < N; i = i + 1 ) {
        for(j = 0; j < vectorSize-1; j = j + 1) {
                if (distance >= horLoc[j] && distance <= horLoc[j+1]){
```

```
                rise = (vertLoc[j]-vertLoc[j+1]);
            run = (horLoc[j] - horLoc[j+1]);
                hypotenuse = pow((pow(rise,2)+pow(run,2)),0.5);
                sinThetas.push_back(-rise/hypotenuse);
                cosThetas.push_back(-run/hypotenuse);
                height = height + dL * sinThetas[i];
                verti.push_back(height);
                distance = distance + dL * cosThetas[i];
                Location.push_back(Location[i] + dL * cosThetas[i]);
                break;
            }
        }
}
/*Where is the wellhead? Find it---*/

WHL = 0;
for(int i = 0; i < N; i = i + 1 ) {
        if (Location[i] > WH_Location){
            WHL = i-1;
            break;
        }
}
```