Computer Program on Heat Transfer – Temperature Distribution – Hydraulic Calculations for Abrasive Jet Drilling

Report

Evren Ozbayoglu

This report describes the basic execution principles and the theoretical background about the computer program developed for Impact Technologies in order to estimate the temperature and pressure distribution inside the wellbore during abrasive drilling using CO_2 and N_2. The report consists of an introduction part, theory part, the algorithm, and "manual" sections.

Introduction

One of the major challenges in drilling operations is to estimate the pressure and temperature distribution inside the wellbore accurately. This is required for assuring correct equipment usage, efficient hole cleaning, maximizing drilling efficiency and minimizing the cost per foot. In conventional drilling operations, the fluid inside the drillstring and annular section can be considered as "same", except the fact that the fluid inside the annulus also contains solid particles due to cuttings and formation contamination, and a negligible amount of formation fluid since usually the conditions are overbalanced, i.e., bottomhole pressure is greater than the formation pressure. However, in abrasive jet drilling, the phase inside the string is single phase (either liquid of supercritical), and after the jet, the fluid shifts to gas phase inside the annulus. Therefore, inside the annulus, there will be gas flowing with a very high velocity, drilled cuttings and some formation fluid since the conditions will be usually underbalanced, i..e., bottomhole pressure is less than the formation pressure.

The temperature and pressure distribution inside a wellbore is a challenging problem, but many work have been conducted and there is a good understanding of the problem when conventional drilling is concerned. For this case, since the fluid is incompressible, the calculation methods are relatively straightforward. However, many approaches available in the literature considers the formation having a constant temperature gradient and related temperature distribution (infinite potential assumption), therefore they ignore the cooling effect of the circulating fluid on the formation around the vicinity of the wellbore. This can be included only by a proper and realistic heat transfer model.

The determination of distribution of temperature and pressure inside the wellbore is much more challenging for abrasive drilling due to the presence of a compressible fluid inside the annulus. Also,

when the fluid passes through the jet and changes phase, a significant temperature drop is observed, which is known as Joule Thompson effect. This phenomenon is not observed for conventional drilling operations.

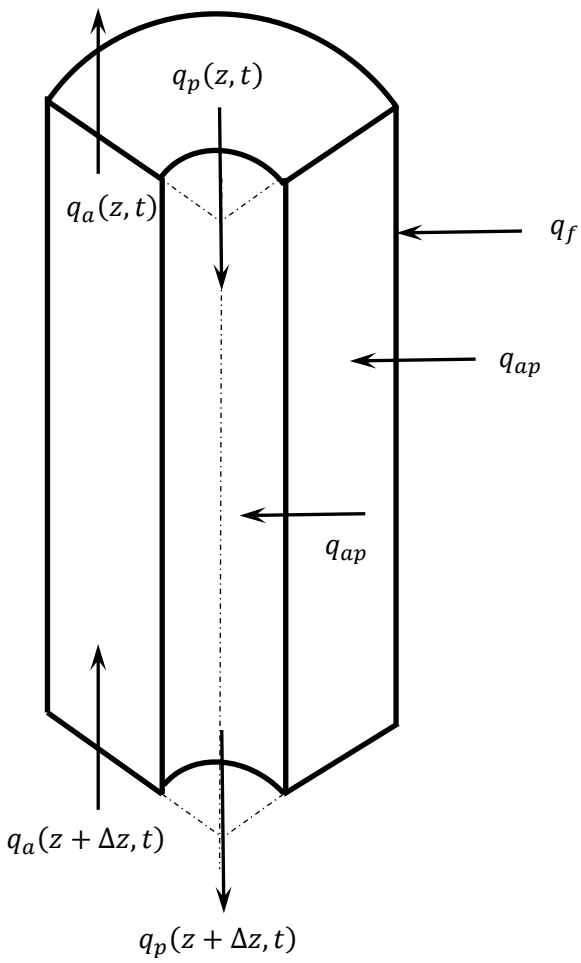
During the scope of this project, a computer program is developed which is capable of estimating temperature and pressure distribution inside the wellbore for both compressible and incompressible fluids considering both Joule Thompson effect and formation cooling.

Theory

This section consists of three parts; i) heat transfer model, ii) hydraulic model, and iii) properties of CO_2.

Heat Transfer Model

In order to determine the temperature distribution inside the wellbore, the following heat transfer model is developed.



According to this model;

The heat rate in drillpipe at z

$q_p(z) = \dot{m}\, c_p\, T_p(z)$

The heat rate in annulus at z

$q_a(z) = \dot{m}\, c_p\, T_a(z)$

The heat rate from annulus to drillpipe

$$q_{ap} = \frac{\dot{m}\, c_p}{B}\left(T_a - T_p\right)\Delta z$$

where

$$B = \frac{\dot{m}\, c_p}{2\,\pi\, r_p\, U_p}$$

The heat transfer from formation to annulus

$$q_{af} = 2\,\pi\, r_w\, U_a (T_w - T_a)\Delta z = \frac{\dot{m}\, c_p}{A}\left(T_f - T_a\right)\Delta z$$

where

$$A = \frac{\dot{m}\, c_p}{2\,\pi\, r_w\, U_a}\left[\frac{k_f + r_w\, U_a\, f(t_D)}{k_f}\right]$$

In order to solve this model numerically, the values of overall heat transfer coefficients must be determined. Overall heat transfer coefficients are

$$\frac{1}{U_p} = \frac{1}{h_p} + \frac{r_{pi}}{k_{st}}\ln\left(\frac{r_{po}}{r_{pi}}\right) + \frac{r_w}{r_{po}}\frac{1}{h_a}$$

between pipe and annulus, and

$$\frac{1}{U_a} = \frac{1}{h_a} + \frac{r_{ci}}{k_{st}}\ln\left(\frac{r_{co}}{r_{ci}}\right) + \frac{r_{ci}}{k_e}\ln\left(\frac{r_{w\prime}}{r_{co}}\right)$$

between annulus and formation. Here, h_p and h_a can be determined by

$$h_p = \left(\frac{k_f}{2\, r_{pi}}\right)\left(\frac{3.65 + \left(0.00668\, N_{Re}\, N_{Pr}\, 2\, r_{pi}\right)}{1 + 0.04\left(N_{Re}\, N_{Pr}\, 2\, r_{pi}\right)^{0.666}}\right) \text{ for } N_{Re} \geq 10{,}000$$

$$h_p = 0.023\, {N_{Re}}^{0.8}\, {N_{Pr}}^{0.4}\, \frac{k_f}{2\, r_{pi}} \text{ for } N_{Re} < 10{,}000$$

$$h_a = \left(\frac{k_f}{2\, r_w}\right)\left(\frac{3.65+(0.00668\, N_{Re}\, N_{Pr}\, 2\, r_w)}{1+0.04\, (N_{Re}\, N_{Pr}\, 2\, r_w)^{0.666}}\right) \text{ for } N_{Re} \geq 10{,}000$$

$$h_a = 0.023\, {N_{Re}}^{0.8}\, {N_{Pr}}^{0.4}\, \frac{k_f}{2\, r_w} \text{ for } N_{Re} < 10{,}000$$

respectively. In these equations, the term N_{Re} is expressed for fluid inside the pipe as

$$N_{Re} = \frac{2\, r_{pi}\, \dot{m}}{A_p\, \mu_p}$$

and for fluid inside the annulus as

$$N_{Re} = \frac{2\, (r_w - r_{po})\, \dot{m}}{A_a\, \mu_p}$$

considering that A_p and A_a are cross-sectional areas of flow for pipe and annulus, respectively. N_{Pr} for both pipe and annulus is defined as

$$N_{Pr} = \frac{c_p\, \mu_p}{k_f}$$

It should be noted that μ_p and k_f values should be defined for both fluid inside the pipe and annulus separately.

The transient heat flow from the formation to the wellbore wall

$$q_f = \frac{2\, \pi\, k_f}{f(t_D)}(T_f - T_a)\Delta z$$

where

$$T_f = T_{sf} + g_T\, z$$

Here, $f(t_D)$ is defined as

$$f(t_D) = 1.1281\sqrt{t_D}\,(1 - 0.3\, t_D) \quad \text{for} \quad 10^{-10} \le t_D \le 1.5$$

and

$$f(t_D) = (0.4063 + 0.5 \ln t_D)\left(1 + \frac{0.6}{t_D}\right) \quad \text{for} \quad t_D > 1.5$$

where

$$t_D = \frac{\alpha_f\, t}{{r_w}^2}$$

and

$$\alpha_f = \frac{k_e}{c_f\, \rho_f}$$

Energy balance applied to the control volume yields;

In drillpipe

$$q_p(z) + q_{ap} = q_p(z + dz)$$

In annulus

$$q_a(z + dz) + q_{af} = q_a(z) + q_{ap}$$

Therefore, using the heat flux definitions and the proper calculus based on the model described, and combining, the governing differential equation can be obtained as

$$A\, B \frac{\partial^2 T_p}{\partial z^2} - B \frac{\partial T_p}{\partial z} - T_p + T_{sf} + g_T\, z$$

General solution of this differential equation which will be used for estimating fluid temperature inside the drillpipe

$$T_p = \gamma\, e^{\varepsilon_1 z} + \delta\, e^{\varepsilon_2 z} + g_T\, z - B\, g_T + T_{sf}$$

and fluid temperature inside the annulus

$$T_a = (1 + \varepsilon_1\, B)\gamma\, e^{\varepsilon_1 z} + (1 + \varepsilon_2\, B)\delta\, e^{\varepsilon_2 z} + g_T\, z + T_{sf}$$

where

$$\varepsilon_1 = \frac{1}{2\,A}\left(1 - \sqrt{1 + \frac{4\,A}{B}}\right)$$

$$\varepsilon_2 = \frac{1}{2\,A}\left(1 + \sqrt{1 + \frac{4\,A}{B}}\right)$$

Applying boundary conditions

$$T_p(z = 0) = T_{inlet}$$

$$\left.\frac{\partial T_p}{\partial z}\right|_{z=D} = c$$

and considering the fact that there will be a temperature reduction due to Joule Thompson effect (sudden expansion) when the fluid changes the phase after the jet;

$$T_p - T_a = \Delta T|_{z=D}$$

the following equation constants are determined;

$$\gamma = -\frac{\left(T_{inlet} + B\ g_T - T_{sf}\right)\varepsilon_2\ e^{\varepsilon_2 D} + g_T + \xi}{\varepsilon_1\ e^{\varepsilon_1 D} - \varepsilon_2\ e^{\varepsilon_2 D}}$$

$$\delta = \frac{\left(T_{inlet} + B\ g_T - T_{sf}\right)\varepsilon_1\ e^{\varepsilon_1 D} + g_T + \xi}{\varepsilon_1\ e^{\varepsilon_1 D} - \varepsilon_2\ e^{\varepsilon_2 D}}$$

where

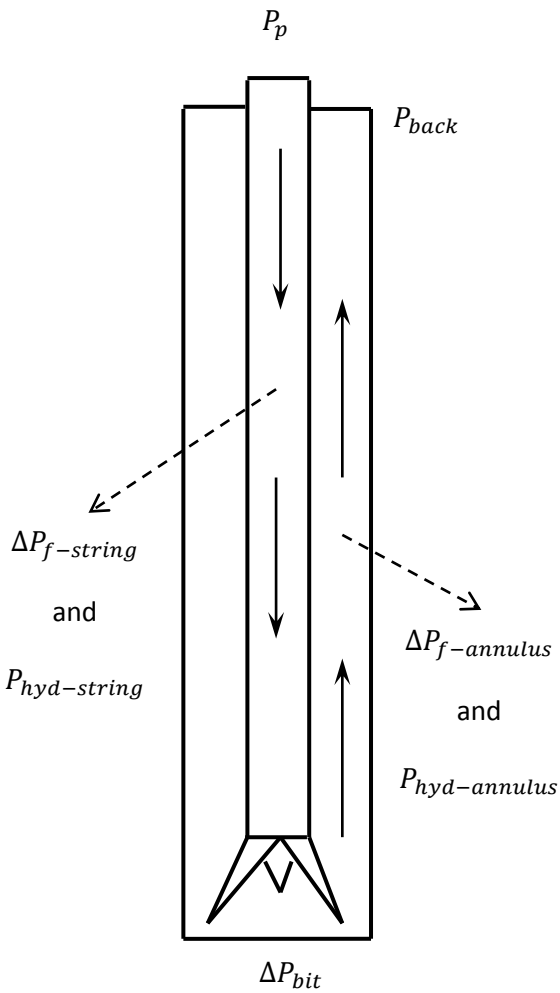
$\xi = f(\Delta T, D)$

Finally, the wellbore wall temperature can be obtained as

$$T_w = B\frac{\partial T_p}{\partial z} + T_p = T_a + \left(T_{sf} + g_T\ z - T_a\right)\frac{\dot{m}\ c_p}{2\ \pi\ r_w\ U_a\ A}$$

<u>Hydraulic Modeling</u>

The following schematic drawing is used to describe the pressure distribution inside the wellbore.



Pump pressure is defined as

$$P_p = \Delta P_{f-string} - P_{hyd-string} + \Delta P_{bit} + \Delta P_{f-annulus} + P_{hyd-annulus} + P_{back}$$

Here,

$$\Delta P_{f-string} = \frac{f_f \, \rho \, v^2}{r_{pi}}$$

$$\Delta P_{f-annulus} = \frac{f_f \, \rho \, v^2}{r_h}$$

where

$$r_h = r_w - r_{po}$$

$$\Delta P_{bit} = \frac{\rho \, Q^2}{2 \, C_d \, {A_n}^2}$$

(C_d is the discharge coefficient, and usually considered as 0.95 for jet nozzles)

$$P_{hyd} = \rho \, g \, D$$

f_f is expressed as a function of N_{Re} using Colebrooks' equation for turbulent flow,

$$\frac{1}{\sqrt{f_f}} = 4 \log\left(N_{Re}\sqrt{f_f}\right) - 0.395$$

and

$$f_f = \frac{16}{N_{Re}}$$

for laminar flow. N_{Re} inside the string is defined as

$$N_{Re} = \frac{2 \, \rho \, v \, r_{pi}}{\bar{\mu}}$$

where

$$\bar{\mu} = \mu_p + \frac{\tau_y \, r_{pi}}{3 \, v}$$

N_{Re} inside the annulus is defined as

$$N_{Re} = \frac{\rho\, v\, 2\, r_h}{\bar{\mu}}$$

where

$$\bar{\mu} = \mu_p + \frac{\tau_y\, r_h}{4\, v}$$

The criteria for laminar to turbulent transition is assumed to be $N_{Re} = 2100$, which is not necessarily to be correct if the fluid has a significant yield stress. However, for this study, the fluid is considered to have no yield stress.

<u>Temperature Drop at the Bit</u>

Due to Joule-Thompson effect, there will be a temperature drop while a compressible fluid flows through a nozzle due to the large pressure drop at the bit. The sudden pressure drop causes an instantaneous volumetric expansion, causing a sudden drop in temperature. Since CO_2 or N_2 will be injected in liquid or supercritical phase through the pipe, while they are passing through the nozzle, the state of phase will change due to pressure drop, which will lead to a temperature decrease. This phenomenon is explained by the general definition of

$$T_2 = \frac{T_1}{\left(\frac{P_1}{P_2}\right)^{\frac{k-1}{k}}}$$

where k is the Joule-Thompson coefficient, defined by the change of pressure with respect to change in temperature at constant enthalpy;

$$k = \left.\frac{\partial P}{\partial T}\right|_H$$

k value shows variations according to the atomic structure of the gas phase, i.e., monoatomic, diatomic and triatomic molecular structures have different *k* values ranging from 1.0 to 1.4.

Influx from Formation

The program can also take the influx from a formation into consideration. Influx causes an increase in the liquid volume inside the annulus, causing an increase in the hydrostatic pressure as well as frictional losses. Also, the temperature distribution inside the wellbore is affected by the influx.

- During this influx consideration, some assumptions are used;
- Productivity index of the formation causing influx is assumed to be constant
- Temperature of the influx is assumed to be equal to the temperature due to geothermal gradient of the field
- Formation pressure is assumed to be equal to the pressure due to normal pressure gradient of the field
- Influx is assumed to be water

The productivity index is defined as

$$PI = \frac{Q}{P_f - P_{ann}}$$

Since annulus may be containing gas, flow rate is converted into mass rate using the in-situ density, which can be determined using real gas law. Total flow inside the annulus becomes the summation of influx rate and volumetric flow rate of the circulation fluid.

Solids in the Wellbore

During the abrasive drilling process, solids are injected through the drillpipe. The contribution of the solids inside the drillpipe is included into consideration as the hydrostatic pressure. The influence of solids inside the pipe on frictional losses is considered only due to density change. The change in the viscosity of the fluid due to presence of solids is ignored.

The mixture density inside the annulus is determined as

$$\rho_{mix} = \rho_f(1 - C_{s1}) + \rho_{s1}C_{s1}$$

Inside the annulus, there are two different solids; i) abrasive solids injected through the drillpipe, and ii) cuttings generated due to drilling process. Therefore, the mixture density inside the annulus considers

both solids types as well as the fluid inside the annulus. As inside the pipe, frictional losses are modified regarding with the change in density due to presence of solids and cuttings inside the annulus.

$$\rho_{mix} = \rho_f\left(1 - (C_{s1+}C_{s2})\right) + \rho_{s1}C_{s1} + \rho_{s2}C_{s2}$$

Additional Water Injection at the Surface

One more option that the program provides is; besides CO_2 or N_2, water can also be injected with these fluids, if needed. The program takes the influence of the additional injected water into consideration for density, viscosity, and fluid specific heat as a mixture. The mixture concentration distribution is determined by using mass rates, i.e., injected liquid has a concentration equal to

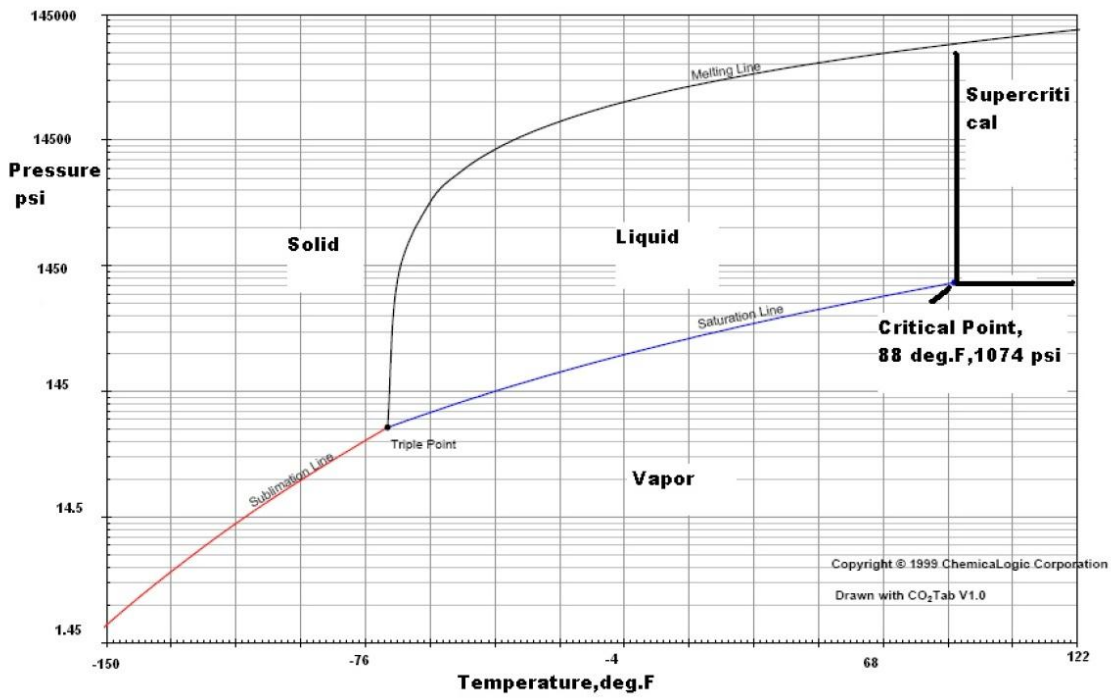
$$C_l = \frac{\dot{m}_l}{\dot{m}_l + \dot{m}_f}$$

CO_2 Properties

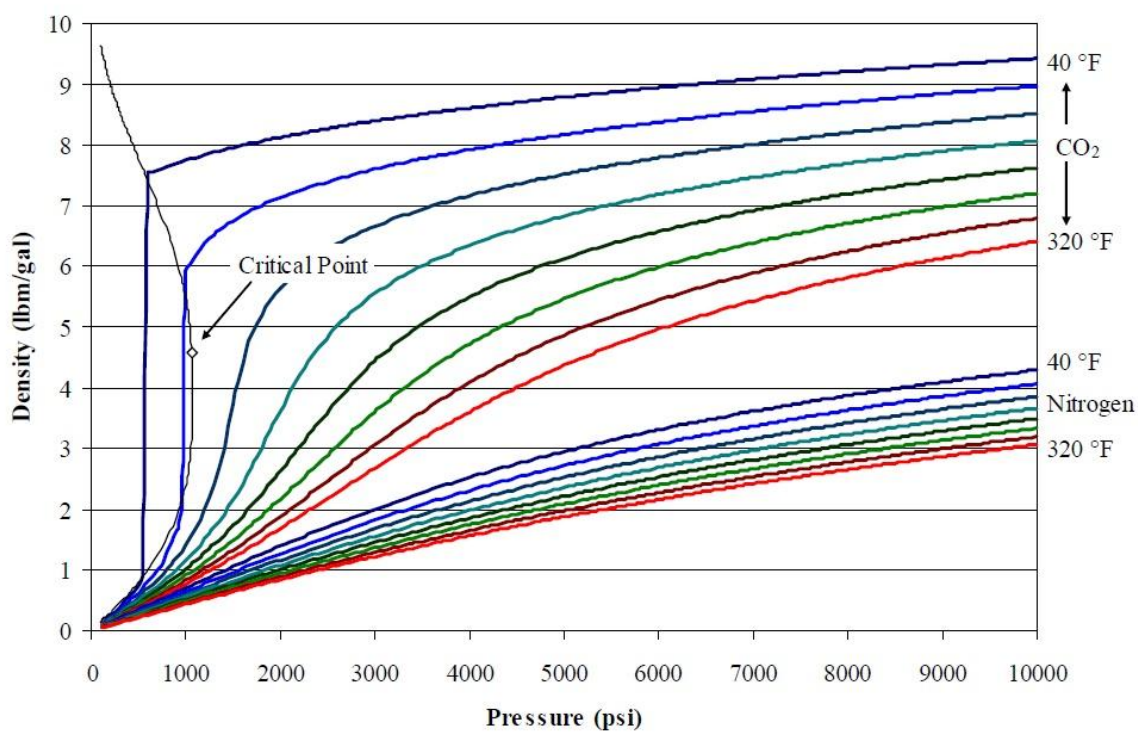
In order to conduct the calculations for temperature and pressure distribution inside the wellbore, the physical properties of CO_2 must be well defined. The properties include the phase, density, viscosity and thermal conductivity.

The following figures are presenting the physical properties of CO_2 as a function of pressure and temperature.

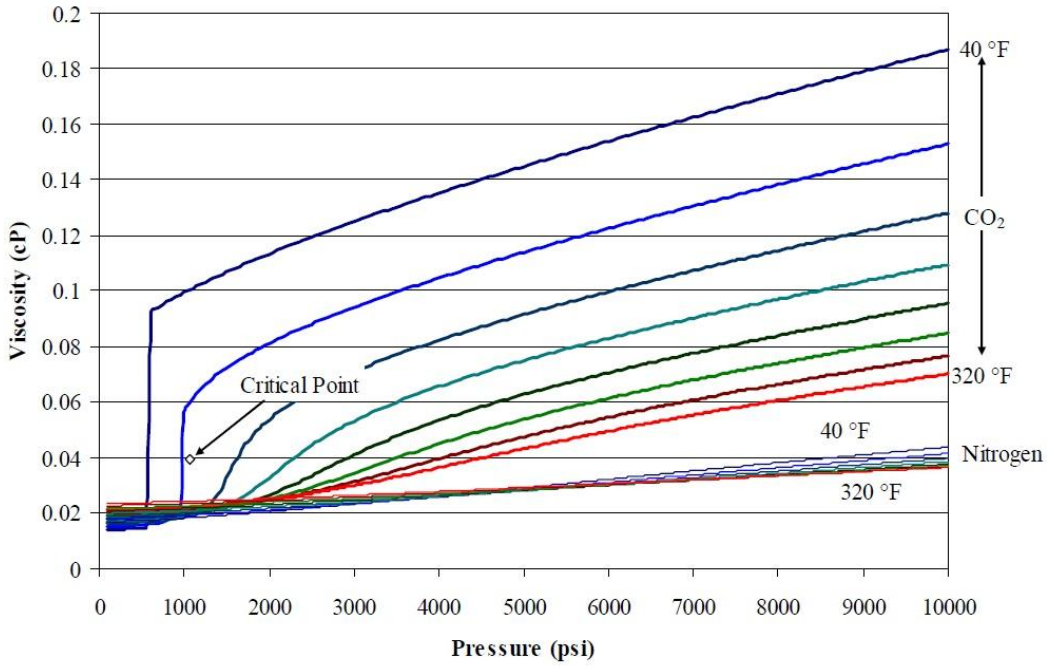
The phase behavior of CO_2 is described as



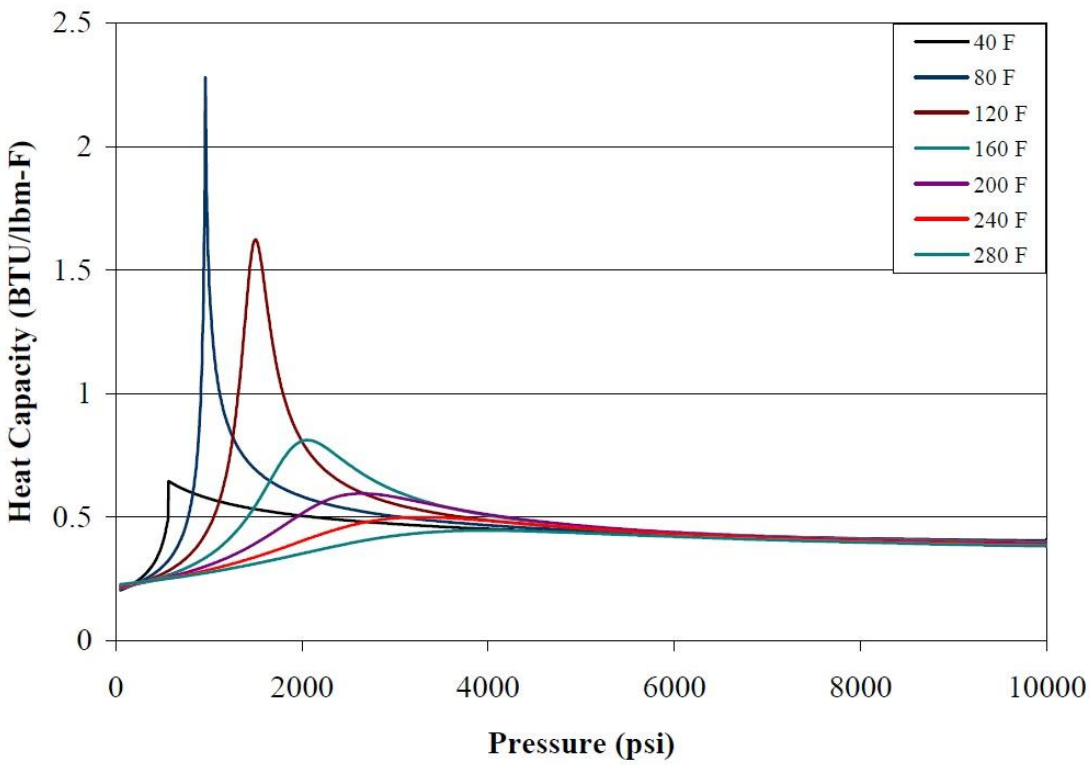
Density of CO_2 is described as shown in the following figure;



Viscosity of CO_2 is given as



Heat capacity of CO_2 is presented in the following figure;



Using the figures presented above, multiple regression techniques are applied after digitizing process for the data given. Then, the following empirical correlations are obtained to define the CO_2 properties;

Density:

$\rho = 4.020134 + 0.000338\,P + 0.000002\,P^2 - 0.04184\,T + 0.000088\,T^2$ for gas phase

$\rho = 6.93583 + 0.00079\,P - 0.036157\,T + 0.000063\,T^2$ for liquid phase and supercritical phase

Viscosity

$\mu = 0.013968\,P^{0.51833}\,T^{-0.548697}$ for gas phase. Since the values are usually around 0.02 cp, this value can also be used as a constant.

$\mu = 0.00000171\,\rho^{4.1878176}$ for liquid phase

$\mu = 1.1\,(0.013968\,P^{0.518333}T^{-0.548697})$ for supercritical phase

Thermal conductivity

$k_f = 0.009$ for gas phase

$k_f = 0.0073\frac{\rho}{7.48} + 0.0036$ for liquid phase

$k_f = 0.015523 + 0.000068\,T^{-0.599136}\,P^{1.0725}$ for supercritical phase

Phase behavior boundaries are approximated using the following equations;

$P_{cr} = 305.0675 + 5.5718\,T + 0.0339\,T^2$

If $> P_{cr}$, then it is liquid phase. Otherwise, the phase is considered as gas.

N_2 Properties

Nitrogen properties are determined using N_2 phase diagrams, and density, viscosity and heat capacity information as a function of temperature and pressure.

Density:

$\rho = 0.318951 + \ 0.000536\, P - 0.003891\, T + 0.000002\, T^2$ for gas phase

$\rho = 0.818951 + 0.000536\, P - 0.003891\, T + 0.000002\, T^2$ for liquid phase and supercritical phase

Viscosity

$\mu = 0.021821627 - 0.00000124\, P + 4.07E - 10\, P^2 - 0.00001512\, T + 1.0307E - 8\, T^2$ for all phases

Thermal conductivity

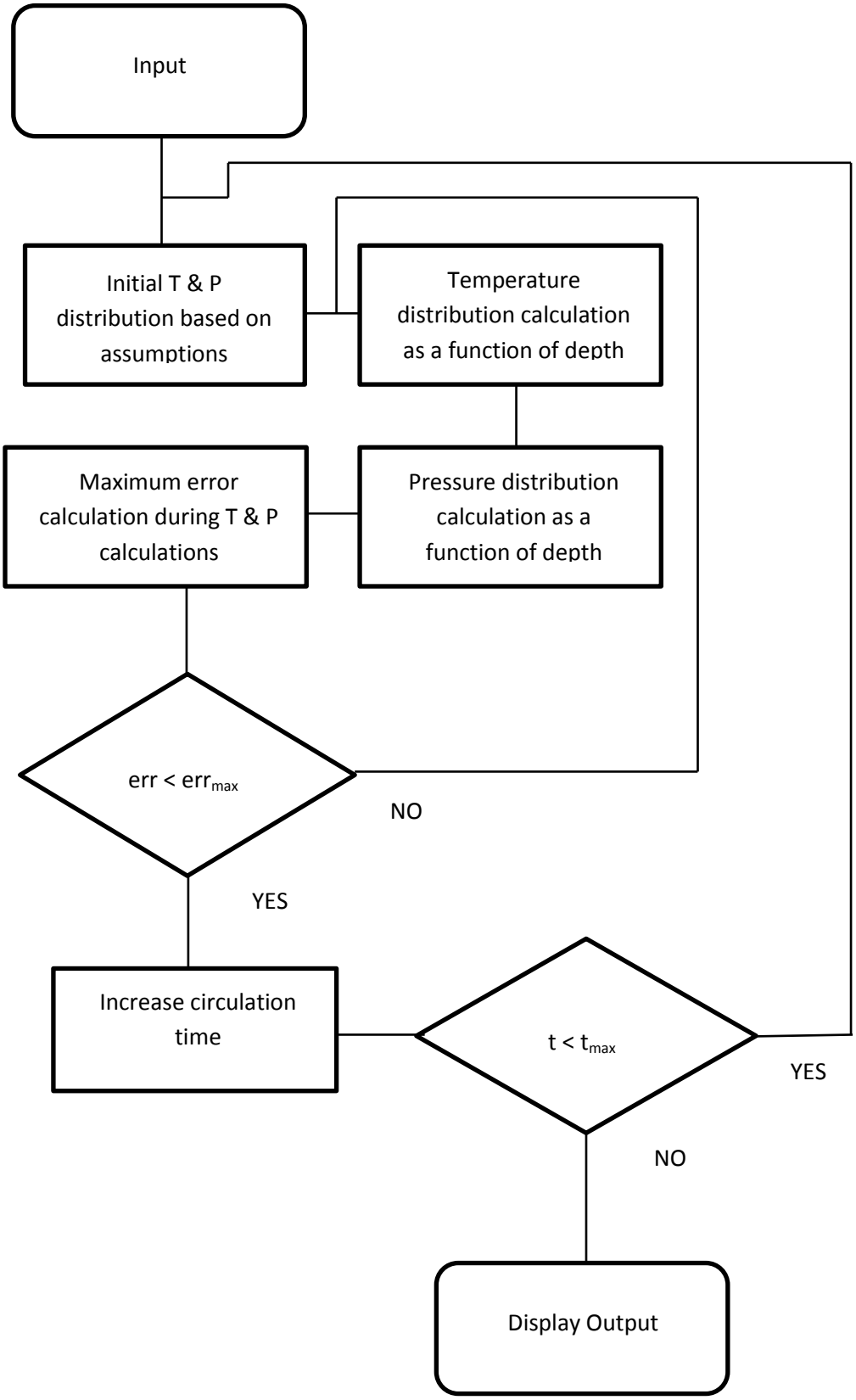
$k_f = 0.01117$ for all phases

Phase behavior boundaries are approximated using the following equations;

$P_{cr} = 8071.4 + 53{,}142\, T + 0.0882\, T^2$

If $> P_{cr}$, then it is liquid phase. Otherwise, the phase is considered as gas.

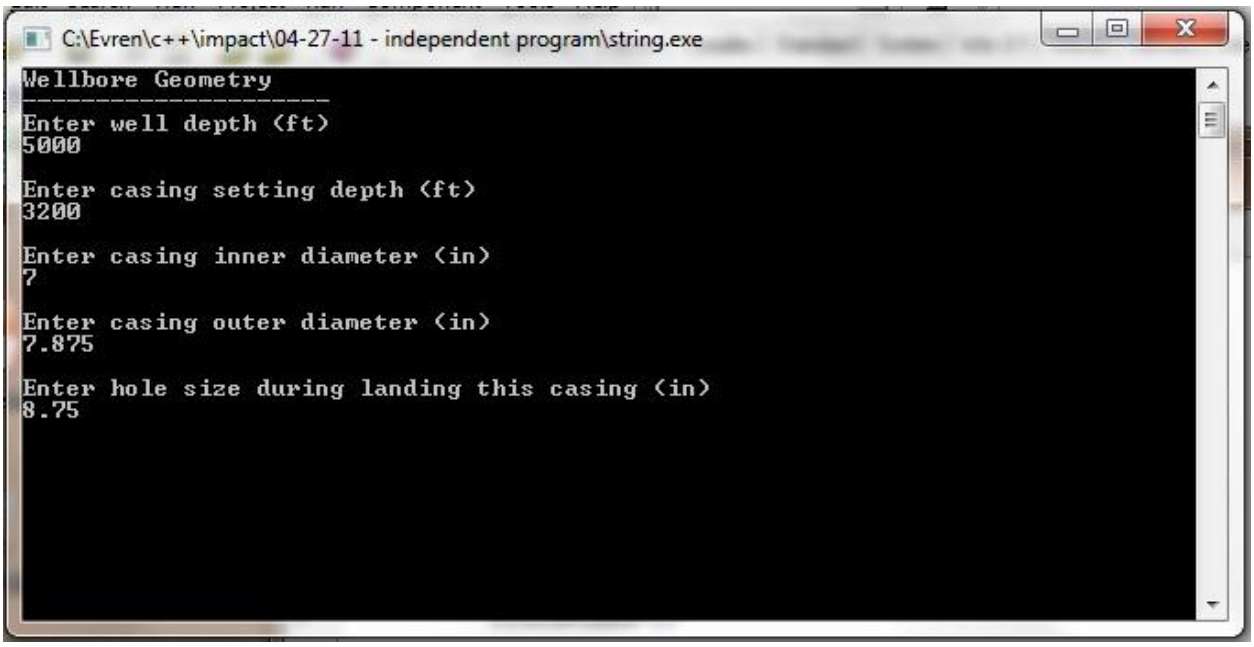
Basic Algorithm of the Computer Program



Manual for the Program

The program has prepared in two different versions; 1) inputs manually entered through the screen by the user, and 2) inputs read from an input file.

First version is prepared to be user friendly while entering the data. Each data required is entered by following the instructions given on the screen, since which data and the required unit is mentioned. A typical input screen is as shown in the figure.



Second version is a direct file input. The input file is shown below:

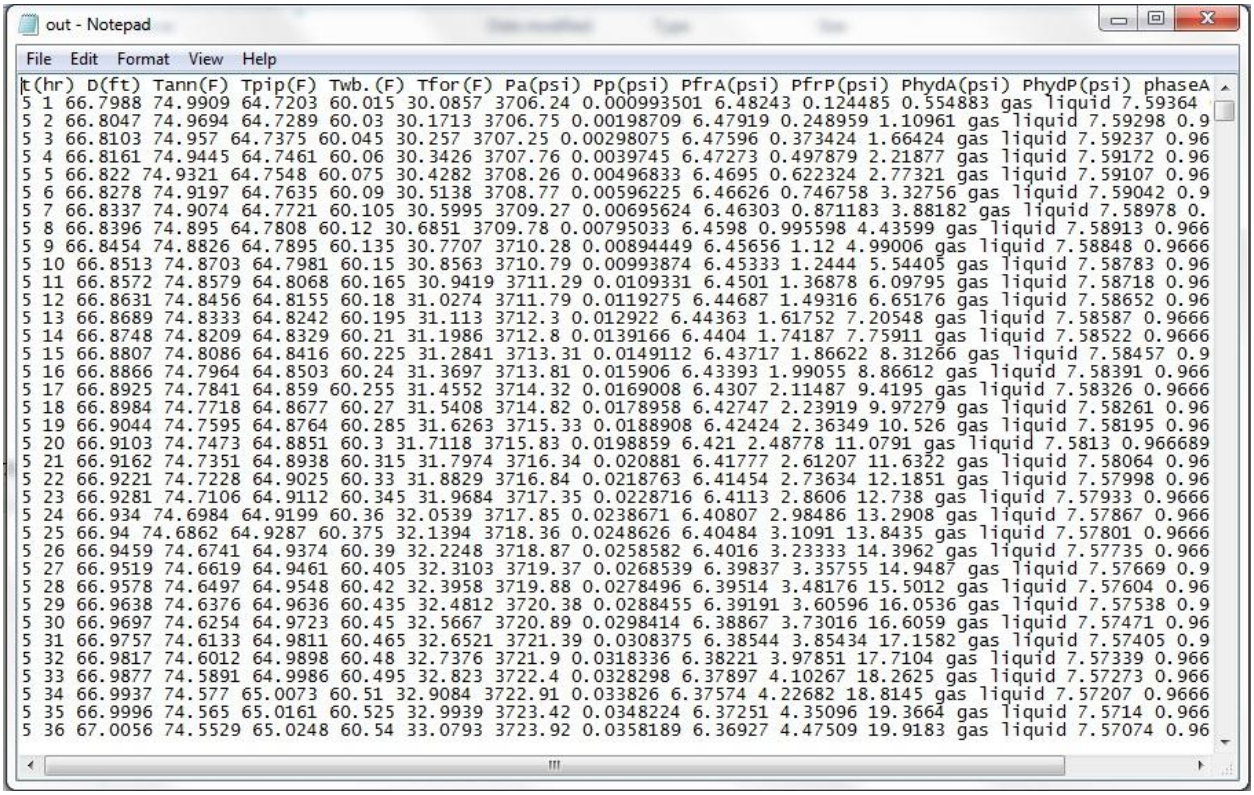
input - Notepad

File Edit Format View Help

```
finalDepth(ft) 5000.
casingSettingDepth(ft) 1000.
casingID(in) 2.25
casingOD(in) 2.5
holeSize(in) 2.5
drillPipeID(in) 1.08
drillPipeOD(in) 1.25
drillCollarID(in) 2.25
drillCollarOD(in) 2.5
drillCollarLength(ft) 0.
pipeThermalCond(BTU/ft-F-hr) 25.
bitDiameter(in) 2.5
numberOfNozzles 1.
nozzleSize(x/32) 1.55
fluidType(1-mud,2-CO2,3-N2) 2
fluidDensity(ppg) 8.33
plasticViscosity(cp) 1.
yieldPoint(lb/100ft^2) 0.
jouleThompsonCoefficient 0.01
flThermalCond(BTU/ft-F-hr) 1.0
fluidSpecificHeat(BTU/lb-F) 0.4
waterDensity(ppg) 8.33
plasticViscosityWater(cp) 1.
yieldPointWater(lb/100ft^2) 0.
jouleThompsonCoefWater 0.
waterTherCond(BTU/ft-F-hr) 1.0
waterSpecificHeat(BTU/lb-F) 0.4
flowRateFluid(gpm) 5.
flowRateWater(gpm) 0.
massRateAbrasives(lb/min) 20.
densityAbrasives(sp.gr.) 2.7
fluidInletTemperature(F) 75.
backPressure(psi) 30.
surfaceTemperature(F) 60.
temperatureGradient(F/100ft) 1.5
formThermalCond(BTU/ft-F-hr) 1.3
formSpecificHeat(BTU/lb-F) 0.2
formationDensity(sp.gr.) 2.64
cuttingsSize(in) 0.001
ROP(ft/hr) 400.
influxDepth(ft) 0.
formationPressGrad(psi/ft) 0.465
prIndex(bbl/day/psi) 0.5
influxDensity(sp.gr) 0.96
```

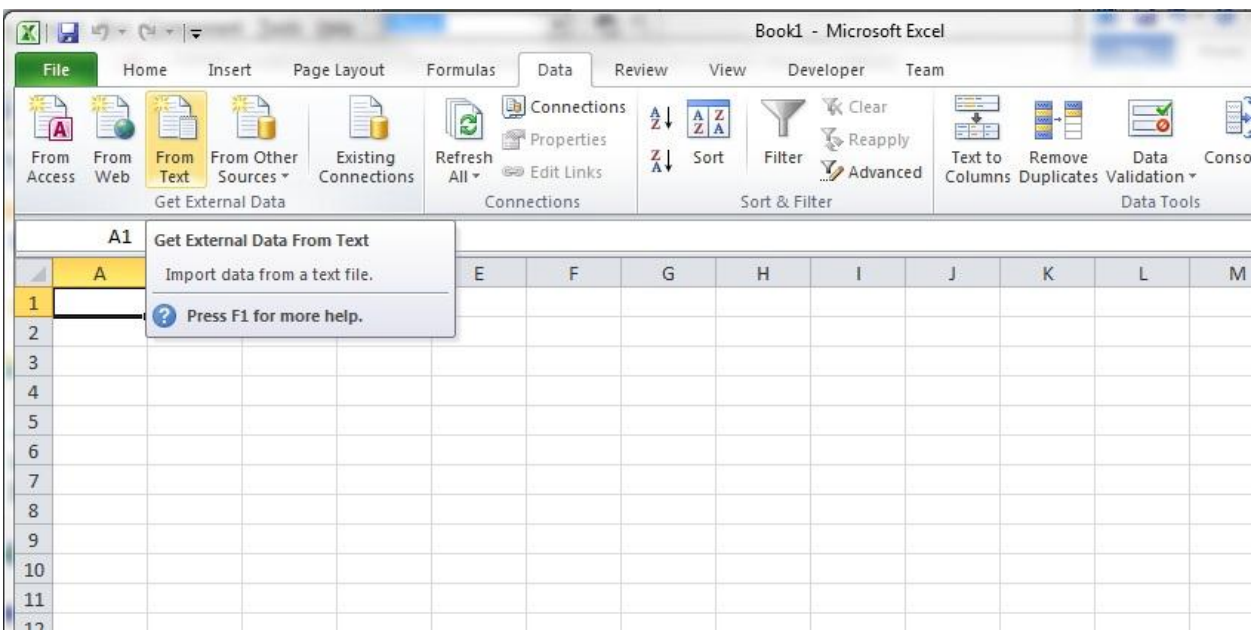
When inputs are completed, the program executes, and prints the results as a table to “out.dat” text file. This file can be directly used in excel by “data import” and “text from file” options. Since the data in the text file is separated by a space from each other, excel will directly recognize, and will put each and every data point in a different cell. Then, any plot can be determined using this data.

A typical out file is shown in the following figure.

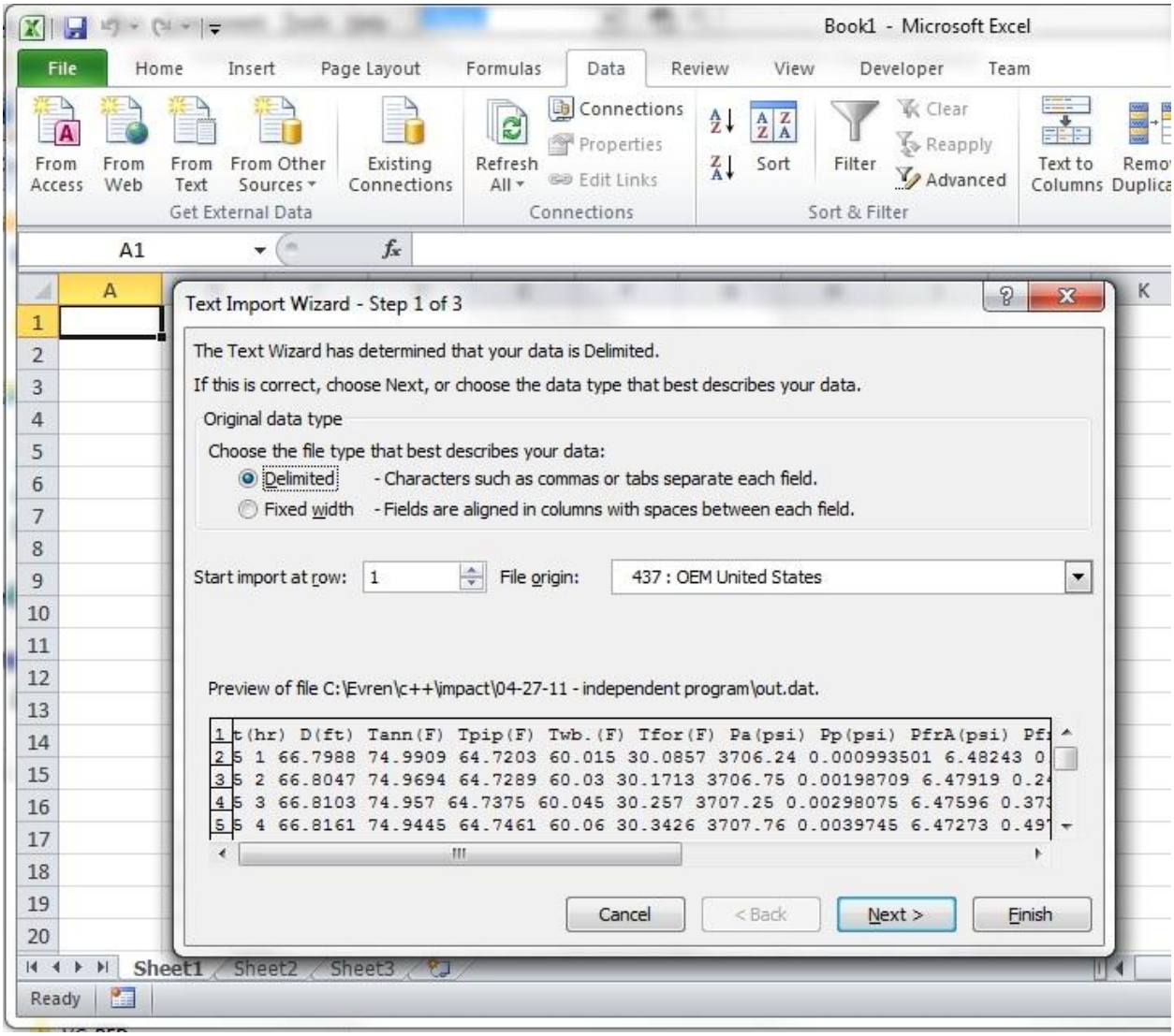


```
t(hr) D(ft) Tann(F) Tpip(F) Twb.(F) Tfor(F) Pa(psi) Pp(psi) PfrA(psi) PfrP(psi) PhydA(psi) PhydP(psi) phaseA
5 1 66.7988 74.9909 64.7203 60.015 30.0857 3706.24 0.000993501 6.48243 0.124485 0.554883 gas liquid 7.59364
5 2 66.8047 74.9694 64.7289 60.03 30.1713 3706.75 0.00198709 6.47919 0.248959 1.10961 gas liquid 7.59298 0.9
5 3 66.8103 74.957 64.7375 60.045 30.257 3707.25 0.00298075 6.47596 0.373424 1.66424 gas liquid 7.59237 0.96
5 4 66.8161 74.9445 64.7461 60.06 30.3426 3707.76 0.0039745 6.47273 0.497879 2.21877 gas liquid 7.59172 0.96
5 5 66.822 74.9321 64.7548 60.075 30.4282 3708.26 0.00496833 6.4695 0.622324 2.77321 gas liquid 7.59107 0.96
5 6 66.8278 74.9197 64.7635 60.09 30.5138 3708.77 0.00596225 6.46626 0.746758 3.32756 gas liquid 7.59042 0.9
5 7 66.8337 74.9074 64.7721 60.105 30.5995 3709.27 0.00695624 6.46303 0.871183 3.88182 gas liquid 7.58978 0.
5 8 66.8396 74.895 64.7808 60.12 30.6851 3709.78 0.00795033 6.4598 0.995598 4.43599 gas liquid 7.58913 0.966
5 9 66.8454 74.8826 64.7895 60.135 30.7707 3710.28 0.00894449 6.45656 1.12 4.99006 gas liquid 7.58848 0.9666
5 10 66.8513 74.8703 64.7981 60.15 30.8563 3710.79 0.00993874 6.45333 1.2444 5.54405 gas liquid 7.58783 0.96
5 11 66.8572 74.8579 64.8068 60.165 30.9419 3711.29 0.0109331 6.4501 1.36878 6.09795 gas liquid 7.58718 0.96
5 12 66.8631 74.8456 64.8155 60.18 31.0274 3711.79 0.0119275 6.44687 1.49316 6.65176 gas liquid 7.58652 0.96
5 13 66.8689 74.8333 64.8242 60.195 31.113 3712.3 0.012922 6.44363 1.61752 7.20548 gas liquid 7.58587 0.9666
5 14 66.8748 74.8209 64.8329 60.21 31.1986 3712.8 0.0139166 6.4404 1.74187 7.75911 gas liquid 7.58522 0.9666
5 15 66.8807 74.8086 64.8416 60.225 31.2841 3713.31 0.0149112 6.43717 1.86622 8.31266 gas liquid 7.58457 0.9
5 16 66.8866 74.7964 64.8503 60.24 31.3697 3713.81 0.015906 6.43393 1.99055 8.86612 gas liquid 7.58391 0.966
5 17 66.8925 74.7841 64.859 60.255 31.4552 3714.32 0.0169008 6.4307 2.11487 9.4195 gas liquid 7.58326 0.9666
5 18 66.8984 74.7718 64.8677 60.27 31.5408 3714.82 0.0178958 6.42747 2.23919 9.97279 gas liquid 7.58261 0.96
5 19 66.9044 74.7595 64.8764 60.285 31.6263 3715.33 0.0188908 6.42424 2.36349 10.526 gas liquid 7.58195 0.96
5 20 66.9103 74.7473 64.8851 60.3 31.7118 3715.83 0.0198859 6.421 2.48778 11.0791 gas liquid 7.5813 0.966689
5 21 66.9162 74.7351 64.8938 60.315 31.7974 3716.34 0.020881 6.41777 2.61207 11.6322 gas liquid 7.58064 0.96
5 22 66.9221 74.7228 64.9025 60.33 31.8829 3716.84 0.0218763 6.41454 2.73634 12.1851 gas liquid 7.57998 0.96
5 23 66.9281 74.7106 64.9112 60.345 31.9684 3717.35 0.0228716 6.4113 2.8606 12.738 gas liquid 7.57933 0.9666
5 24 66.934 74.6984 64.9199 60.36 32.0539 3717.85 0.0238671 6.40807 2.98486 13.2908 gas liquid 7.57867 0.966
5 25 66.94 74.6862 64.9287 60.375 32.1394 3718.36 0.0248626 6.40484 3.1091 13.8435 gas liquid 7.57801 0.9666
5 26 66.9459 74.6741 64.9374 60.39 32.2248 3718.87 0.0258582 6.4016 3.23333 14.3962 gas liquid 7.57735 0.966
5 27 66.9519 74.6619 64.9461 60.405 32.3103 3719.37 0.0268539 6.39837 3.35755 14.9487 gas liquid 7.57669 0.9
5 28 66.9578 74.6497 64.9548 60.42 32.3958 3719.88 0.0278496 6.39514 3.48176 15.5012 gas liquid 7.57604 0.96
5 29 66.9638 74.6376 64.9636 60.435 32.4812 3720.38 0.0288455 6.39191 3.60596 16.0536 gas liquid 7.57538 0.9
5 30 66.9697 74.6254 64.9723 60.45 32.5667 3720.89 0.0298414 6.38867 3.73016 16.6059 gas liquid 7.57471 0.96
5 31 66.9757 74.6133 64.9811 60.465 32.6521 3721.39 0.0308375 6.38544 3.85434 17.1582 gas liquid 7.57405 0.9
5 32 66.9817 74.6012 64.9898 60.48 32.7376 3721.9 0.0318336 6.38221 3.97851 17.7104 gas liquid 7.57339 0.966
5 33 66.9877 74.5891 64.9986 60.495 32.823 3722.4 0.0328298 6.37897 4.10267 18.2625 gas liquid 7.57273 0.966
5 34 66.9937 74.577 65.0073 60.51 32.9084 3722.91 0.033826 6.37574 4.22682 18.8145 gas liquid 7.57207 0.9666
5 35 66.9996 74.565 65.0161 60.525 32.9939 3723.42 0.0348224 6.37251 4.35096 19.3664 gas liquid 7.5714 0.966
5 36 67.0056 74.5529 65.0248 60.54 33.0793 3723.92 0.0358189 6.36927 4.47509 19.9183 gas liquid 7.57074 0.96
```

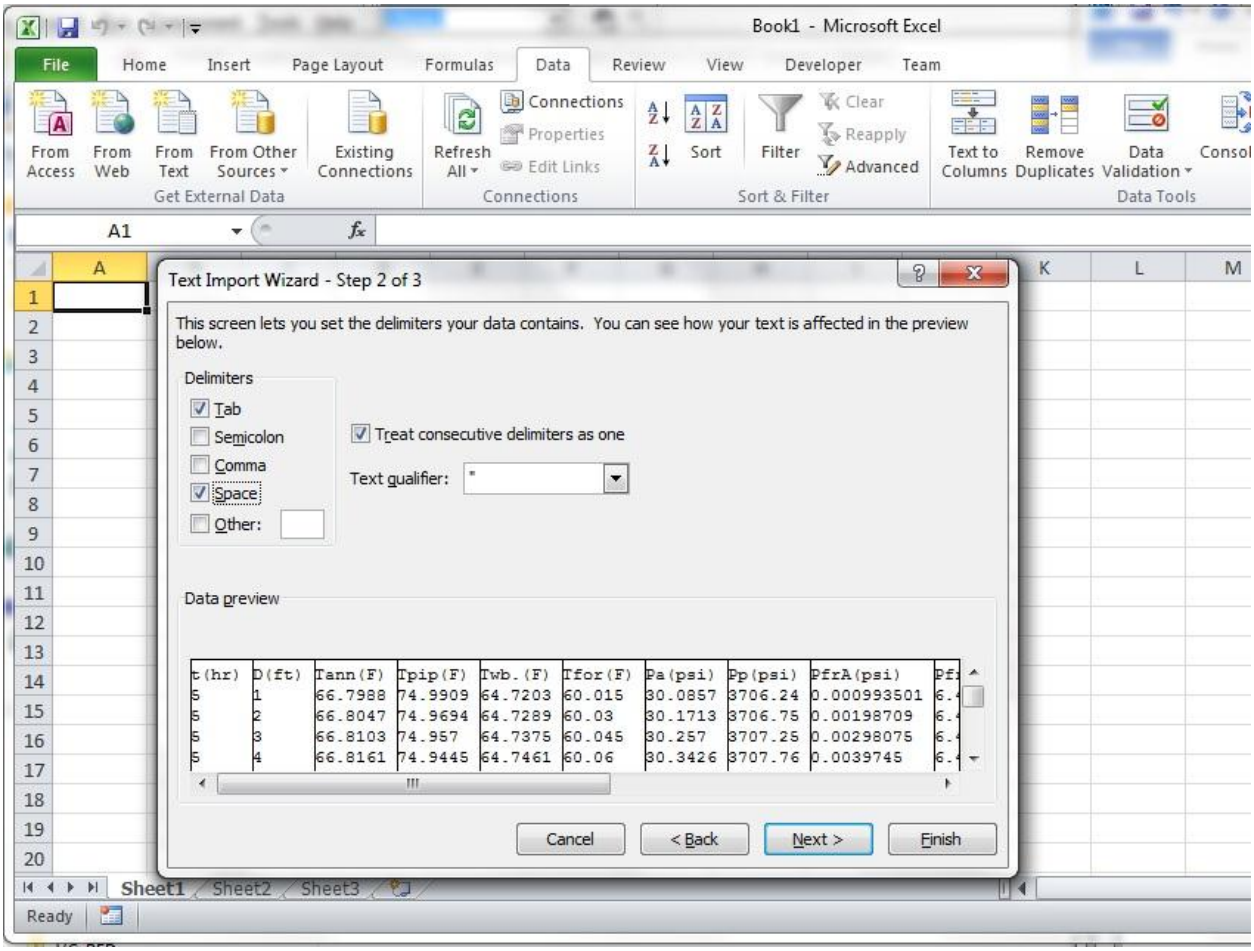
This file is transferred to excel using the “data import” utility.



After selecting the file “out.dat” from the folder that it is generated, steps should be followed.



In the “delimiters” box, “space” should be marked.



After pressing “Finish” all the data will be transferred onto excel such that, each data point will be recorded on a separate cell.

In case of a failure in calculation during the program is running, an error message will pop up, mentioning that the inputs should be modified.

Nomenclature

α	heat diffusivity of formation
A_p	cross-sectional area of inside of the drillpipe

A_a	cross-sectional area of inside of the annulus
C_d	jet discharge coefficent
c_f	specific heat of formation
ΔP_f	frictional pressure losses
ΔP_{bit}	pressure loss at the bit
D	total depth
c_p	specific heat of drilling fluid
f_f	friction factor
g	gravitational constant
g_T	geothermal gradient
h_a	coefficient of heat transfer of fluid in annulus
h_p	coefficient of heat transfer of fluid in drillpipe
D	well depth
k_{st}	thermal conductivity of pipe or casing
k_e	thermal conductivity of formation and/or cement
k_f	thermal conductivity of drilling fluid
$\dot{m}$	mass flow rate
μ_p	plastic viscosity
$\bar{\mu}$	effective viscosity
N_{Pr}	Prandtl number
N_{Rep}	Reynolds number for drillpipe
N_{Rea}	Reynolds number for annulus
P_{hyd}	hydrostatic pressure
Q	volumetric flow rate
q_{ap}	conductive heat flow across drillpipe
q_{af}	conductive heat flux from formation
q_a	convective heat flow in the annulus

q_p	convective heat flow in the drillpipe, BTU/hr
ρ_f	density of formation, lb/gal
ρ	density of drilling fluid, lb/gal
r_{pi}	drillpipe inner radius
r_{po}	drillpipe outer radius
r_{ci}	casing inner radius
r_{co}	casing outer radius
r_w	wellbore radius
t	circulation time
τ_y	yield stress
T_{inlet}	drillpipe inlet fluid temperature
T_{sf}	surface earth temperature
T_w	temperature at wellbore-formation interface
T_a	fluid temperature in the annulus
T_p	fluid temperature in the drillpipe
T_f	temperature of the formation
t_D	dimensionless time
U_p	overall heat transfer coefficient from annulus to drillpipe
U_a	overall heat transfer coefficient from formation to annulus
v	velocity

Subscripts

g	gas phase
l	liquid phase
mix	mixture
1,2	upstream, downstream conditions
s	Solid

APPENDIX

C++ source code for the software developed

```
//---------------------------------------------------------------------------

#include <vcl.h>
#include <conio.h>
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>
#include <stdlib.h>
#include <math.h>
#pragma hdrstop

//---------------------------------------------------------------------------


// variable declarations

int choice, fluidType, fail;
const double e=2.718281828, pi=3.141592654;

// depth - wellbore configuration
double finalDepth;
double casingSettingDepth, casingID, casingOD, holeSize;

// drillstring properties
double drillPipeOD, drillPipeID;
double drillCollarOD, drillCollarID, drillCollarLength;
double pipeThermalConductivity;
// double pipeSpecificHeat;

// bit information
double bitDiameter, numberOfNozzles, nozzleSize;

// fluid properties
double fluidDensity, plasticViscosity, yieldPoint;
double waterDensity, plasticViscosityWater, yieldPointWater;
double fluidThermalConductivity, fluidSpecificHeat, fluidSpecificHeatPipe;
double waterThermalConductivity, waterSpecificHeat;
double overallHeatTransferCoefficientPipeAnnulus;
double overallHeatTransferCoefficientAnnulusFormation;
double flowRate, massFlowRate;
double massFlowRateFluid, massFlowRateWater;
double flowRateFluid, flowRateWater;
double fluidInletTemperature;
double jouleThompsonCoefficient, jouleThompsonCoefficientWater, cPovercV;
double ha, hp;

// abrasive solids
double massRateAbrasives, densityAbrasives;

// pressure - temperature info
double backPressure;
```

```
double surfaceTemperature, temperatureGradient;

// formation properties
double formationThermalConductivity, formationSpecificHeat, formationDensity,
cuttingsSize;

// influx properties
double influxRate, influxDepth, formationPressureGradient, prIndex, influxMassRate,
influxDensity;

// other variables
double circulationTime, circulationTime2, dcirculationTime, finalCirculationTime, depth,
dDepth;
double temperatureInsidePipe[2][20000], temperatureInsideAnnulus[2][20000],
temperatureWellbore[2][20000], temperatureFormation[20000], tBottomhole;
double tfModified, tD;
double pressureInsidePipe[2][20000], pressureInsideAnnulus[2][20000];
double frictionAnnulus[20000], frictionPipe[20000], hydrostaticPipe[20000],
hydrostaticAnnulus[20000];
double hydrostaticPressure, frictionalPressureLoss, bitPressureLoss;
double stringID[20000], stringOD[20000], casID[20000], casOD[20000], boreSize[20000],
boreSizeOld[20000];
double cementThermalConductivity;
double bitUpstreamPressure, bitDownstreamPressure, bitPressureDrop,
bitUpstreamTemperature, bitDownstreamTemperature;
double maxErr, err, printMaxErr;
double deltaT, delT[20000];
double cuttingsConcentration[20000], ROP, transportRatio;
int fluidPhase, phaseInsidePipe[20000], phaseInsideAnnulus[20000];
int errDepth;
bool printErr, screenInput;

// temporary parameters, can be removed later
double parUa[20000], parUp[20000], parK1[20000], parK2[20000];


void clearScreen()
{
 // cleans the screen
     getch();
     clrscr();

}



void inputPage()
{

  bool check;
  int morePipe,i,pipeNo;
  char defDum1[15],defDum2[23],defDum3[13],defDum4[13],defDum5[13];
  char defDum6[16],defDum7[16],defDum8[18],defDum9[18],defDum10[22];
  char defDum11[29],defDum12[16],defDum13[16],defDum14[17],defDum15[28];
  char defDum16[18],defDum17[21],defDum18[23],defDum19[25],defDum20[27];
  char defDum21[28],defDum22[18],defDum23[26],defDum24[28],defDum25[23];
  char defDum26[27],defDum27[28],defDum28[19],defDum29[19],defDum30[26];
  char defDum31[25],defDum32[25],defDum33[18],defDum34[22],defDum35[29];
```

```
 char defDum36[29],defDum37[27],defDum38[25],defDum39[17],defDum40[11];
 char defDum41[16],defDum42[27],defDum43[21],defDum44[21];
//   clrscr();

//   printf("Please enter the input type\n");
//   printf("\n");
//   printf("For manual input, press '0', then press ENTER\n");
//   printf("For file input, press '1', then press ENTER\n");
//   printf("\n");

//   cin >> screenInput;

   screenInput=1;
   switch (screenInput)
   {
     case 0: // Manual Input
     {

       printf("Input Data:\n");
       printf("\n");
       printf("(Press any key to continue ...)\n");
       getch();
       clrscr();

       printf("Wellbore Geometry\n");
       printf("---------------------\n");
       printf("Enter well depth (ft)\n");
       cin >> finalDepth;
       printf("\n");

       printf("Enter casing setting depth (ft)\n");
       cin >> casingSettingDepth;
       printf("\n");

       if (casingSettingDepth>0)
       {
         printf("Enter casing inner diameter (in)\n");
         cin >> casingID;
         printf("\n");

         printf("Enter casing outer diameter (in)\n");
         cin >> casingOD;
         printf("\n");

         printf("Enter hole size during landing this casing (in)\n");
         cin >> holeSize;
         printf("\n");
       }
       clrscr();

       printf("Drillstring Properties\n");
       printf("---------------------\n");
       printf("Enter drillpipe inner diameter (in)\n");
       cin >> drillPipeID;
       printf("\n");

       printf("Enter drillpipe outer diameter (in)\n");
       cin >> drillPipeOD;
```

```
        printf("\n");

        printf("Enter drill collar length (ft)\n");
        cin >> drillCollarLength;
        printf("\n");

        if (drillCollarLength>0)
        {
          printf("Enter drill collar inner diameter (in)\n");
          cin >> drillCollarID;
          printf("\n");

          printf("Enter drill collar outer diameter (in)\n");
          cin >> drillCollarOD;
          printf("\n");
        }

        printf("Enter thermal conductivity of the pipe (BTU/ft-F-hr)\n");
        cin >> pipeThermalConductivity;
        printf("\n");

//        printf("Enter specific heat of the pipe (BTU/ft-F-hr)\n");
//        cin >> pipeSpecificHeat;
//        printf("\n");

        clrscr();

        printf("Bit Properties\n");
        printf("----------------------\n");
        printf("Enter bit diameter (in)\n");
        cin >> bitDiameter;
        printf("\n");

        printf("Enter number of nozzles \n");
        cin >> numberOfNozzles;
        printf("\n");

        printf("Enter nozzle size \n");
        cin >> nozzleSize;
        printf("\n");

        clrscr();

        printf("Fluid Properties\n");
        printf("----------------------\n");
        printf("Select fluid type\n");
        printf(" \n");
        printf("1 - Mud\n");
        printf("2 - CO2\n");
        printf("3 - N2\n");
        printf(" \n");
        printf("(please select the fluid type, then press ENTER\n");
        cin >> fluidType;
        clrscr();

        switch (fluidType)
        {
          case 1:
```

```
      {
        printf("Enter mud density (ppg)\n");
        cin >> fluidDensity;
        printf("\n");

        printf("Enter plastic viscosity (cp)\n");
        cin >> plasticViscosity;
        printf("\n");

        printf("Enter yield point (lb/100ft^2)\n");
        cin >> yieldPoint;
        printf("\n");

        printf("Enter thermal conductivity of the fluid (BTU/ft-F-hr)\n");
        cin >> fluidThermalConductivity;
        printf("\n");

        printf("Enter specific heat of the fluid (BTU/lb-F)\n");
        cin >> fluidSpecificHeat;
        printf("\n");

        clrscr();
        break;
      }

      case 2:
      {
// CO2 properties are calculated in the program (these are dummy values)
        fluidDensity=1.;
        plasticViscosity=1.;
        yieldPoint=1.;
        fluidThermalConductivity=1.;
        fluidSpecificHeat=1.;

        break;
      }

      case 3:
      {
// N2 properties are calculated in the program (these are dummy values)
        fluidDensity=1.;
        plasticViscosity=1.;
        yieldPoint=1.;
        fluidThermalConductivity=1.;
        fluidSpecificHeat=1.;

        break;
      }
    }

    printf("Enter flow rate (gpm)\n");
    cin >> flowRateFluid;
    printf("\n");

    printf("Enter flow rate of water injected (gpm) (if no, enter 0)\n");
    cin >> flowRateWater;
    printf("\n");
```

```
        printf("Enter fluid inlet temperature (F)\n");
        cin >> fluidInletTemperature;
        printf("\n");

        printf("Enter backpressure (psi)\n");
        cin >> backPressure;
        printf("\n");

        clrscr();

        printf("Abrasive Solids Information\n");
        printf("----------------------\n");
        printf("Enter mass rate of abrasive solids (lb/min)\n");
        cin >> massRateAbrasives;
        printf("\n");

        printf("Enter density of abrasive solids (sp.gr)\n");
        cin >> densityAbrasives;
        printf("\n");

        clrscr();

        printf("Temperature Information\n");
        printf("----------------------\n");
        printf("Enter surface temperature (F)\n");
        cin >> surfaceTemperature;
        printf("\n");

        printf("Enter temperature gradient (F/100ft)\n");
        cin >> temperatureGradient;
        printf("\n");

        clrscr();

        printf("Formation Properties\n");
        printf("----------------------\n");
        printf("Enter thermal conductivity of the formation (BTU/ft-F-hr)\n");
        cin >> formationThermalConductivity;
        printf("\n");

        printf("Enter specific heat of the formation (BTU/lb-F)\n");
        cin >> formationSpecificHeat;
        printf("\n");

        printf("Enter formation density (sp.gr.)\n");
        cin >> formationDensity;
        printf("\n");

        printf("Enter influx depth (ft) (if no flux, set depth to 0)\n");
        cin >> influxDepth;
        printf("\n");

        if (influxDepth>0)
        {
          printf("Enter productivity index for influx (bbl/day/psi) (suggested value =
0.5)\n");
          cin >> prIndex;
          printf("\n");
```

```
          printf("Enter formation pressure gradient (psi/ft) (if not known, enter
0.465)\n");
          cin >> formationPressureGradient;
          printf("\n");

          printf("Enter influx density (sp.gr) (if no flux, set depth to 0)\n");
          cin >> influxDensity;
          printf("\n");
        }


        printf("Enter average cuttings size (in)\n");
        cin >> cuttingsSize;
        printf("\n");

        printf("Enter rate of penetration (ft/hr)\n");
        cin >> ROP;
        printf("\n");


        clrscr();

       break;
      }

      case 1:  // File input
      {
        ifstream inFile;
        inFile.open ("input.dat");
        inFile >> defDum1 >> finalDepth;
        inFile >> defDum2 >> casingSettingDepth;
        inFile >> defDum3 >> casingID;
        inFile >> defDum4 >> casingOD;
        inFile >> defDum5 >> holeSize;
        inFile >> defDum6 >> drillPipeID;
        inFile >> defDum7 >> drillPipeOD;
        inFile >> defDum8 >> drillCollarID;
        inFile >> defDum9 >> drillCollarOD;
        inFile >> defDum10 >> drillCollarLength;
        inFile >> defDum11 >> pipeThermalConductivity;

        inFile >> defDum12 >> bitDiameter;
        inFile >> defDum13 >> numberOfNozzles;
        inFile >> defDum14 >> nozzleSize;

        inFile >> defDum15 >> fluidType;
        inFile >> defDum16 >> fluidDensity;
        inFile >> defDum17 >> plasticViscosity;
        inFile >> defDum18 >> yieldPoint;
        inFile >> defDum19 >> jouleThompsonCoefficient;
        inFile >> defDum20 >> fluidThermalConductivity;
        inFile >> defDum21 >> fluidSpecificHeat;

        inFile >> defDum22 >> waterDensity;
        inFile >> defDum23 >> plasticViscosityWater;
        inFile >> defDum24 >> yieldPointWater;
        inFile >> defDum25 >> jouleThompsonCoefficientWater;
```

```
        inFile >> defDum26 >> waterThermalConductivity;
        inFile >> defDum27 >> waterSpecificHeat;
        inFile >> defDum28 >> flowRateFluid;
        inFile >> defDum29 >> flowRateWater;

        inFile >> defDum30 >> massRateAbrasives;
        inFile >> defDum31 >> densityAbrasives;

        inFile >> defDum32 >> fluidInletTemperature;
        inFile >> defDum33 >> backPressure;
        inFile >> defDum34 >> surfaceTemperature;
        inFile >> defDum35 >> temperatureGradient;
        inFile >> defDum36 >> formationThermalConductivity;
        inFile >> defDum37 >> formationSpecificHeat;
        inFile >> defDum38 >> formationDensity;
        inFile >> defDum39 >> cuttingsSize;
        inFile >> defDum40 >> ROP;

        inFile >> defDum41 >> influxDepth;
        inFile >> defDum42 >> formationPressureGradient;
        inFile >> defDum43 >> prIndex;
        inFile >> defDum44 >> influxDensity;


        inFile.close();

//      clrscr();
      break;
      }
    }
/*
  finalDepth=5.;        // 15000
  casingSettingDepth=0.;  // 10000
  casingID=2.25;
  casingOD=2.5;
  holeSize=2.5;
  drillPipeID=1.08;
  drillPipeOD=1.25;
  drillCollarID=2.25;
  drillCollarOD=2.5;
  drillCollarLength=0.;
  pipeThermalConductivity=25;
//  pipeSpecificHeat=600;
  bitDiameter=2.5;
  numberOfNozzles=1;
  nozzleSize=1.55;
  fluidType=2.;
  fluidDensity=8.33;
  plasticViscosity=1.;
  yieldPoint=0;
  jouleThompsonCoefficient=0.01;
  fluidThermalConductivity=1.0;    // 0.33
  fluidSpecificHeat=0.4;          // 1.1

  waterDensity=8.33;
  plasticViscosityWater=1.;
  yieldPointWater=0;
  jouleThompsonCoefficientWater=0.;
```

```
  waterThermalConductivity=1.0;    // 0.33
  waterSpecificHeat=0.4;          // 1.1

  flowRateFluid=5.;
  flowRateWater=0.;

  massRateAbrasives=20.;
  densityAbrasives=2.7;

  fluidInletTemperature=75.;
  backPressure=30.;
  surfaceTemperature=60.;
  temperatureGradient=1.5;
  formationThermalConductivity=1.3;
  formationSpecificHeat=0.2;
  formationDensity=2.64;
  cuttingsSize=0.001;
  ROP=400.;

  influxDepth=0.;
  formationPressureGradient=0.465;
  prIndex=0.5;
  influxDensity=0.96;
*/

  if (influxDepth == 0)
  {
    influxDensity=8.;
    prIndex=0.1;
    formationPressureGradient=0.465;
    waterDensity=8.33;
    plasticViscosityWater=1.;
    yieldPointWater=0;
    jouleThompsonCoefficientWater=0.;
  }

  if (casingSettingDepth == 0)
  {
    casingID=bitDiameter-1.;
    casingOD=bitDiameter;
    holeSize=bitDiameter;
  }

  if (drillCollarLength == 0)
  {
    drillCollarID=drillPipeID;
    drillCollarOD=drillPipeOD;
  }

}


void unitConversion()
{
 // converting units to proper field units for calculations (if needed)

//  finalDepth    // ft
//  casingSettingDepth  // ft
```

```
  casingID=casingID/12.;   // in to ft
  casingOD=casingOD/12.;   // in to ft
  holeSize=holeSize/12.;   // in to ft
  drillPipeID=drillPipeID/12.;  // in to ft
  drillPipeOD=drillPipeOD/12.;  // in to ft
  drillCollarID=drillCollarID/12.;   // in to ft
  drillCollarOD=drillCollarOD/12.;   // in to ft
//  drillCollarLength    // ft
//  pipeThermalConductivity  // BTU-ft/(ft^2-hr-F)
//  pipeSpecificHeat     //  BTU/(lb-F)
  bitDiameter=bitDiameter/12.;  // in to ft
  nozzleSize=nozzleSize/32.;  // in*32 to in
  fluidDensity=fluidDensity*7.48;  // lb/gal to lb/ft^3
  plasticViscosity=plasticViscosity*2.4191;   // cp to lb/(ft-hr)
  yieldPoint=yieldPoint*100.;   // lb/100-ft^2 to lb/ft^2
  waterDensity=waterDensity*7.48;  // lb/gal to lb/ft^3
  influxDensity=influxDensity*62.4;  // sp.gr to lb/ft^3
  plasticViscosityWater=plasticViscosityWater*2.4191;   // cp to lb/(ft-hr)
  yieldPointWater=yieldPointWater*100.;   // lb/100-ft^2 to lb/ft^2
//  fluidThermalConductivity    // BTU-ft/(ft^2-hr-F)
//  fluidSpecificHeat    //  BTU/(lb-F)
  flowRateFluid=flowRateFluid*8.020833;  // gpm to ft^3/hr
  flowRateWater=flowRateWater*8.020833;  // gpm to ft^3/hr
  backPressure=backPressure*144.;   // psi to lb/ft^2
  massRateAbrasives=massRateAbrasives*60.;  // lb/min to lb/hr
  densityAbrasives=densityAbrasives*62.4;  // sp.gr to lb/ft^3
  formationPressureGradient=formationPressureGradient*144.;
//  surfaceTemperature   // F
//  surfaceTemperature   // F
  temperatureGradient=temperatureGradient/100.;  // F/100-ft to F/ft
//  formationThermalConductivity   // BTU-ft/(ft^2-hr-F)
//  formationSpecificHeat    //   BTU/(lb-F)
  formationDensity=formationDensity*62.4;   // sp.gr. to lb/ft^3
  cuttingsSize=cuttingsSize/12.;          // in to ft
  prIndex=prIndex*0.0016247;       // bbl/day/psi to ft^3/hr/lb/ft^2
// ROP     // ft/hr

}


void outputPage()
{
 //   printf("Results written on 'out.dat' file \n");
 //   printf(" \n");

    if (printErr==1)
    {
 //     printf("\n");
 //     printf("Forced convergence : ");
 //     cout << printMaxErr;
 //     printf(" at ");
 //     cout << errDepth;
 //     printf (" ft");
 //     printf("\n");
 //     printf("\n");
    }

 //   printf("Press any key to continue ...");
```

```
}


void intro()
{

 //   printf("Welcome to Impact Design Program\n");
 //   printf("\n");
 //   printf("Press any key to continue ..\n");
 //   clearScreen();

}


void intro1()
{

 //   printf("Please press '1' to continue, '0' for EXIT, then press ENTER\n");
 //   printf("\n");

}


void fluidPropertyDetermination(double t, double p, double s)
{
  double phaseP;

 // this section requires modification according to the fluid in use

    if (p<=0.)
     p=0.001;

    switch (fluidType)
    {
      case 1:
      {
        fluidPhase=0;

        fluidDensity=fluidDensity;

        plasticViscosity=plasticViscosity;
        yieldPoint=yieldPoint;

        fluidThermalConductivity=fluidThermalConductivity;
        fluidSpecificHeat=fluidSpecificHeat;

        jouleThompsonCoefficient=jouleThompsonCoefficient;
        cPovercV=1.;

        break;
      }

      case 2:     // CO2
      {
       // phase calculation
        phaseP=305.0675+5.5718*t+0.0339*pow(t,2.);
        if (p<phaseP) // gas phase
        {
```

```
          fluidPhase=1;
 //         if (t<=0)
           fluidDensity=4.020134+p*0.000338+0.000002*pow(p,2.)-
0.041840*t+0.000088*pow(t,2.);
 //         else
 //          fluidDensity=0.00238*pow(p,1.82064)*pow(t,-1.18085);
          if (fluidDensity<0.05)
           fluidDensity=0.05;
          else if (fluidDensity>10.)
           fluidDensity=10.;
          fluidDensity=fluidDensity*7.48;

 //         plasticViscosity=0.013968*pow(p,0.518333)*pow(t,-0.548697);
 //         if (plasticViscosity<0.0001)
 //          plasticViscosity=0.0001;
          plasticViscosity=0.02;
          plasticViscosity=plasticViscosity*2.4191;
          yieldPoint=0.0012;

       //   fluidThermalConductivity=0.015523+0.000068*pow(t,-0.599136)*pow(p,1.0725);
          fluidThermalConductivity=0.009;
          fluidSpecificHeat=0.201961+0.0000803237057561171*t-1.4522474376118E-
08*pow(t,2.)+8.06809443440048E-13*pow(t,3.);

       //  jouleThompsonCoefficient=0.0012; // F/psi
          jouleThompsonCoefficient=0.012;
          cPovercV=1.37;
        }

        if (p>=phaseP) // liquid phase
        {
          fluidPhase=0;

      //    fluidDensity=1.389954*pow(p,0.356736)*pow(t,-0.311397);      //
fluidDensity=-0.0007*pow(t,2.)+0.0323*t+7.3848;
          fluidDensity=6.935830+p*0.000790+0.000000*pow(p,2.)-
0.036157*t+0.000063*pow(t,2.);
          if (fluidDensity>12.)
           fluidDensity=12.;

          plasticViscosity=0.0000171*pow(fluidDensity,4.18781759);     //
plasticViscosity=-0.00007*pow(t,2.)-0.0012*t+1.1116;     //
plasticViscosity=0.013968*pow(p,0.518333)*pow(t,-0.548697);
          if (plasticViscosity<0.0001)
           plasticViscosity=0.0001;
          plasticViscosity=plasticViscosity*2.4191;
          fluidDensity=fluidDensity*7.48;
          yieldPoint=0.;

      //    fluidThermalConductivity=-0.000004*pow(t,2.)+0.0001*t+0.0613;        //
fluidThermalConductivity=0.015523+0.000068*pow(t,-0.599136)*pow(p,1.0725);
          fluidThermalConductivity=0.0073*fluidDensity/7.48+0.0036;
      //    fluidSpecificHeat=0.497;
          fluidSpecificHeat=195645.*pow((fluidDensity/7.48),-6.306);

       //   jouleThompsonCoefficient=0.0012; // F/psi
          jouleThompsonCoefficient=0.0012;
          cPovercV=1.05;
```

```
        }

        if ((p>1054.)&&(t>87.65)) // supercritical phase
        {
          fluidPhase=2;

          // fluidDensity=1.389954*pow(p,0.356736)*pow(t,-0.311397);
          fluidDensity=6.935830+p*0.000790+0.000000*pow(p,2.)-
0.036157*t+0.000063*pow(t,2.);
          if (fluidDensity>10.)
           fluidDensity=10.;

          fluidDensity=fluidDensity*7.48;
          plasticViscosity=1.1*(0.013968*pow(p,0.518333)*pow(t,-0.548697));
          if (plasticViscosity<0.0001)
           plasticViscosity=0.0001;
          plasticViscosity=plasticViscosity*2.4191;
          yieldPoint=0.;

          fluidThermalConductivity=0.015523+0.000068*pow(t,-0.599136)*pow(p,1.0725);
       //   fluidSpecificHeat=0.201961+0.0000803237057561171*t-1.4522474376118E-
08*pow(t,2.)+8.06809443440048E-13*pow(t,3.);
          fluidSpecificHeat=0.497;

       //   jouleThompsonCoefficient=0.0012; // F/psi
          jouleThompsonCoefficient=0.0015;
          cPovercV=1.05;
        }


        break;
      }

      case 3:     // N2
      {
       // phase calculation
        phaseP=0.0882*pow(t,2.)+53.142*t+8071.4;
        if (p<phaseP) // gas phase
        {
          fluidPhase=1;
          if (t<0)
           fluidDensity=0.318951+p*0.000536-0.000000*pow(p,2.)-
0.003891*t+0.000002*pow(t,2.);
          else
           fluidDensity=0.023887*pow(p,0.662326)*pow(t,-0.212517);
          if (fluidDensity<0.01)
           fluidDensity=0.01;
          else if (fluidDensity>10.)
           fluidDensity=10.;
          fluidDensity=fluidDensity*7.48;

          plasticViscosity=0.021821627261426+p*-1.24207108829138E-06+4.07329587745606E-
10*pow(p,2.)+t*-0.0000151200634793847+1.30689389694566E-08*pow(t,2.);
          if (plasticViscosity<0.0001)
           plasticViscosity=0.0001;
          plasticViscosity=plasticViscosity*2.4191;
          yieldPoint=0.;
```

```
       //   fluidThermalConductivity=0.015523+0.000068*pow(t,-0.599136)*pow(p,1.0725);
          fluidThermalConductivity=0.01117;
          fluidSpecificHeat=0.231961+0.0000803237057561171*t-1.4522474376118E-
08*pow(t,2.)+8.06809443440048E-13*pow(t,3.);

       //  jouleThompsonCoefficient=0.0012; // F/psi
          jouleThompsonCoefficient=0.012;
          cPovercV=1.39;
        }

        if (p>=phaseP) // liquid phase
        {
       //   fluidDensity=0.023887*pow(p,0.662326)*pow(t,-0.212517);
          fluidDensity=0.818951+p*0.000536-0.000000*pow(p,2.)-
0.003891*t+0.000002*pow(t,2.);
          if (fluidDensity<0.05)
           fluidDensity=0.05;
          else if (fluidDensity>12.)
           fluidDensity=12.;
          fluidDensity=fluidDensity*7.48;

          plasticViscosity=0.021821627261426+p*-1.24207108829138E-06+4.07329587745606E-
10*pow(p,2.)+t*-0.0000151200634793847+1.30689389694566E-08*pow(t,2.);
          if (plasticViscosity<0.0001)
           plasticViscosity=0.0001;
          plasticViscosity=plasticViscosity*2.4191;
          yieldPoint=0.;

       //   fluidThermalConductivity=0.015523+0.000068*pow(t,-0.599136)*pow(p,1.0725);
          fluidThermalConductivity=0.01117;
          fluidSpecificHeat=0.231961+0.0000803237057561171*t-1.4522474376118E-
08*pow(t,2.)+8.06809443440048E-13*pow(t,3.);

       //   jouleThompsonCoefficient=0.0012; // F/psi
          jouleThompsonCoefficient=0.0012;
          cPovercV=1.39;
        }

        if ((p>1054.)&&(t>87.65)) // supercritical phase
        {
          fluidPhase=2;
          fluidDensity=0.818951+p*0.000536-0.000000*pow(p,2.)-
0.003891*t+0.000002*pow(t,2.);
      //    fluidDensity=0.023887*pow(p,0.662326)*pow(t,-0.212517);
          if (fluidDensity<0.05)
           fluidDensity=0.05;
          else if (fluidDensity>10.)
           fluidDensity=10.;
          fluidDensity=fluidDensity*7.48;

          plasticViscosity=0.021821627261426+p*-1.24207108829138E-06+4.07329587745606E-
10*pow(p,2.)+t*-0.0000151200634793847+1.30689389694566E-08*pow(t,2.);
          if (plasticViscosity<0.0001)
           plasticViscosity=0.0001;
          plasticViscosity=plasticViscosity*2.4191;
          yieldPoint=0.;

       //   fluidThermalConductivity=0.015523+0.000068*pow(t,-0.599136)*pow(p,1.0725);
```

```
          fluidThermalConductivity=0.01117;
          fluidSpecificHeat=0.231961+0.0000803237057561171*t-1.4522474376118E-
08*pow(t,2.)+8.06809443440048E-13*pow(t,3.);

       //   jouleThompsonCoefficient=0.0012; // F/psi
          jouleThompsonCoefficient=0.0015;
          cPovercV=1.39;
        }

        break;
      }
    }

}


void fluidPropertyMixture()
{
  double flowRateF, flowRateW, flowRateT;

    flowRateF=massFlowRateFluid/fluidDensity;
    flowRateW=massFlowRateWater/waterDensity;
    flowRateT=flowRateF+flowRateW;

    fluidDensity=fluidDensity*(flowRateF/flowRateT)+waterDensity*(flowRateW/flowRateT);

plasticViscosity=plasticViscosity*(flowRateF/flowRateT)+plasticViscosityWater*(flowRateW/
flowRateT);
    yieldPoint=yieldPoint*(flowRateF/flowRateT)+yieldPointWater*(flowRateW/flowRateT);

fluidThermalConductivity=fluidThermalConductivity*(flowRateF/flowRateT)+waterThermalCondu
ctivity*(flowRateW/flowRateT);

fluidSpecificHeat=fluidSpecificHeat*(flowRateF/flowRateT)+waterSpecificHeat*(flowRateW/fl
owRateT);

jouleThompsonCoefficient=jouleThompsonCoefficient*(flowRateF/flowRateT)+jouleThompsonCoef
ficientWater*(flowRateW/flowRateT);

    flowRate=flowRateT;
}


void overallHeatTransferEstimation(int k)
{
  double rpi, rpo, kp, rci, rco, rb, rbo, ka, ke, kt;
  double nRep, nRea, nPrp, nPra, aP, aA, mP, mA, cpP, cpA;
  double temp, pres;
  double mixF, mixI, mFR;
  int infD;

  // definitions
    infD=influxDepth;
    mFR=massFlowRate;
    kt=pipeThermalConductivity;
    ke=formationThermalConductivity;

    rpi=stringID[k]/2.;
```

```
    rpo=stringOD[k]/2.;

    rci=casID[k]/2.;
    rco=casOD[k]/2.;
    rb=boreSize[k]/2.;
    rbo=boreSizeOld[k]/2.;

    aP=pi*pow(rpi,2.);
    aA=pi*(pow(rb,2.)-pow(rpo,2.));

   // modified pipe temperature
    if (k==1)
     temp=temperatureInsidePipe[0][k];
    else
     temp=(temperatureInsidePipe[0][k]+temperatureInsidePipe[0][k-1])/2.;

    pres=pressureInsidePipe[0][k];
    fluidPropertyDetermination(temp, pres/144., k);
    fluidPropertyMixture();

   // pipe
    mP=plasticViscosity;
    kp=fluidThermalConductivity;
    cpP=fluidSpecificHeat;
    fluidSpecificHeatPipe=cpP;


   // modified annulus temperature
    if (k==1)
     temp=temperatureInsideAnnulus[0][k];
    else
     temp=(temperatureInsideAnnulus[0][k]+temperatureInsideAnnulus[0][k-1])/2.;

    pres=pressureInsideAnnulus[0][k];
    fluidPropertyDetermination(temp, pres/144., k);
    fluidPropertyMixture();

   // annulus
    ka=fluidThermalConductivity;
    cpA=fluidSpecificHeat;
    mA=plasticViscosity;


   // influx contribution
    influxMassRate=prIndex*(formationPressureGradient*influxDepth-
pressureInsideAnnulus[0][infD])*influxDensity;

mixF=(massFlowRate/fluidDensity)/((massFlowRate/fluidDensity)+(influxMassRate/influxDensi
ty));

mixI=(influxMassRate/influxDensity)/((massFlowRate/fluidDensity)+(influxMassRate/influxDe
nsity));

    if (k<=infD)
    {
      if ((formationPressureGradient*influxDepth)>pressureInsideAnnulus[0][infD])
      {
        temp=temp*mixF+temperatureFormation[infD]*mixI;
```

```
        pres=pressureInsideAnnulus[0][k];
        fluidPropertyDetermination(temp, pres/144., k);
        fluidPropertyMixture();
        mA=mixF*plasticViscosity+mixI*plasticViscosityWater;
        mFR=mFR+influxMassRate;
      }
    }


   // NReP, NReA, NPrP and NPrA calculations
    nRep=(2.*rpi*massFlowRate)/(aP*mP);
    nRea=(0.816*2.*(rb-rpo)*mFR)/(aA*mA);

    nPrp=cpP*mP/kp;
    nPra=cpA*mA/ka;

   // Heat Transfer Coefficients, ha and hp
    if (nRep<=10000.)

hp=(kp/(2.*rpi))*(3.65+(0.00668*nRep*nPrp*(2.*rpi))/(1+0.04*(pow((nRep*nPrp*2.*rpi),0.666
))));
    else
     hp=0.023*pow(nRep,0.8)*pow(nPrp,0.4)*kp/(2.*rpi);   // N/s-m-K

    if (nRea<=10000.)

ha=(ka/(2.*rb))*(3.65+(0.00668*nRea*nPra*(2.*rb))/(1+0.04*(pow((nRea*nPra*2.*rb),0.666)))
);
    else
     ha=0.023*pow(nRea,0.8)*pow(nPra,0.4)*ka/(2.*rb);    // N/s-m-K

   // Overall heat transfer coefficients, Ua and Up

overallHeatTransferCoefficientPipeAnnulus=1/((1/hp)+(rpi/kt)*log(rpo/rpi)+(rb/rpo)*(1/ha)
);   // N/s-m-K
//    overallHeatTransferCoefficientPipeAnnulus=30.;

overallHeatTransferCoefficientAnnulusFormation=1/((1/ha)+(rci/kt)*log(rco/rci)+(rci/ke)*l
og(rbo/rco));   // N/s-m-K
//    overallHeatTransferCoefficientAnnulusFormation=1.0;
}


void tfCalculation(double d)
{
  int i;

  // no modification is conducted at this moment
    i=d;

tfModified=(2.*(surfaceTemperature+temperatureGradient*d)+0.*temperatureWellbore[0][i])/2
.;
}



void formationTemperatureDetermination(int d, double ta, double tf)
{
```

```
  double kf, cf, rb, ua, alfa, ttd, df;

     overallHeatTransferEstimation(d);

     ua=overallHeatTransferCoefficientAnnulusFormation;

     kf=formationThermalConductivity;
     cf=formationSpecificHeat;
     df=formationDensity;
     rb=boreSize[d]/2.;

     tfCalculation(d);

     tf=(tf+tfModified)/2.;    // tfModified=tf

     alfa=kf/(cf*df);
     ttd=alfa*circulationTime/pow(rb,2.);   // dimensionless

     if (ttd<=1.5)
      tD=1.1281*pow(ttd,0.5)*(1.-0.3*pow(ttd,0.3));
     else
      tD=(0.4063+0.5*log(ttd))*(1.+0.6/ttd);

     if (d==1)
      temperatureWellbore[1][d]=(kf*tf+rb*ua*tD*ta)/(kf+rb*ua*tD);
     else
      temperatureWellbore[1][d]=(kf*tf+rb*ua*tD*ta)/(kf+rb*ua*tD);

 //    temperatureWellbore[1][d]=ta+(surfaceTemperature+(temperatureGradient-
deltaT/finalDepth)*d-ta)*(kf/(kf+rb*ua*tD));
     temperatureWellbore[1][d]=ta+(surfaceTemperature+(temperatureGradient)*d-
ta)*(kf/(kf+rb*ua*tD));

}


void temperatureUpdate()
{
  double ta, tp, tf;
  int i;


    i=finalDepth;
    ta=temperatureInsideAnnulus[1][i];
    tf=surfaceTemperature+temperatureGradient*i;
    formationTemperatureDetermination(i, ta, tf);

    for (i=finalDepth-1; i>=1; i--)
    {
      ta=(temperatureInsideAnnulus[1][i]+temperatureInsideAnnulus[1][i+1])/2.;
      tf=surfaceTemperature+temperatureGradient*i;
      formationTemperatureDetermination(i, ta, tf);
    }

}
```

```
void temperatureCalculation(int j, double tf)
{
  double a, b;
  double ta, ta1, ta2, tp, tp1, tp2;
  double c1, c2, c3, c4, k1, k2;
  double tpC1, tpC2, taC1, taC2;
  double rpi, uP;
  double rb, uA;
  double tpInlet, ts, tg, h;
  double ke, eps1, eps2, gama, delta;
  double dt, epsilon;
  double mFRT, mixI, mixF;
  double gamaDum,gama1,delta1;
  double failDum1, failDum2;
  int fd, inD;


    tpInlet=fluidInletTemperature;
    ts=surfaceTemperature;
    tg=temperatureGradient;
    h=finalDepth;
    fd=h;
    ke=formationThermalConductivity;
    inD=influxDepth;

    overallHeatTransferEstimation(j);
    uP=overallHeatTransferCoefficientPipeAnnulus;
    uA=overallHeatTransferCoefficientAnnulusFormation;

    rb=boreSize[j]/2.;
    rpi=stringID[j]/2.;

  // influx influence
    influxMassRate=prIndex*(formationPressureGradient*influxDepth-
pressureInsideAnnulus[0][inD])*influxDensity;
 //
mixF=(massFlowRate/fluidDensity)/((massFlowRate/fluidDensity)+(influxMassRate/influxDensi
ty));
 //
mixI=(influxMassRate/influxDensity)/((massFlowRate/fluidDensity)+(influxMassRate/influxDe
nsity));

   // influx contribution
    mFRT=0.;

    if (j<=inD)
    {
     if ((formationPressureGradient*influxDepth)>pressureInsideAnnulus[0][inD])
      {
        mFRT=influxMassRate;
      }
    }


  // temperature calculation methodology - I

    a=((massFlowRate+mFRT)*fluidSpecificHeatPipe/(2.*pi))*((ke+tD)/(rb*uA*ke));       //
lb/hr
```

```
    b=(massFlowRate*fluidSpecificHeatPipe/(2.*pi*rpi*uP));                          //
ft

    eps1=1./(2.*a)+1./(2.*a)*pow((1.+4.*a/b),0.5);
    eps2=1./(2.*a)-1./(2.*a)*pow((1.+4.*a/b),0.5);

    failDum1=eps1;
    if (failDum1<0)
     failDum1=-failDum1;
    failDum2=eps2;
    if (failDum2<0)
     failDum2=-failDum2;

    if ((failDum1>0.1)||(failDum2>0.1))
    {
      clrscr();
      printf("Unstable conditions !! .. \n");
      printf("\n");
      printf("Please change inputs (increase nozzle size, reduce flow rate, etc)\n");
      printf("\n");
      fail=1;
      eps1=0.00000001;
      eps2=-0.00000001;
      gama1=1.;
      delta1=1.;
    }

    if (fail==0)
     gama1=(eps1*pow(e,(eps1*fd))-eps2*pow(e,(eps2*fd)));
    gama=-((tpInlet+b*(tg)-ts)*eps2*pow(e,(eps2*fd))+tg+deltaT/fd*3.)/gama1;
//    delta=((tpInlet+b*(tg)-
ts)*eps1*pow(e,(eps1*fd))+tg+deltaT/fd)/(eps1*pow(e,(eps1*fd))-eps2*pow(e,(eps2*fd)));
//    gamaDum=deltaT/fd;
//    gama=gamaDum-((tpInlet+b*(tg)-ts)*eps2*pow(e,(eps2*fd))+tg)/(eps1*pow(e,(eps1*fd))-
eps2*pow(e,(eps2*fd)));
//    gamaDum=deltaT/fd*(eps1*pow(e,(eps1*fd))-
eps2*pow(e,(eps2*fd))+eps1*pow(e,(eps1*fd))*eps2*pow(e,(eps2*fd)))/(eps1*pow(e,(eps1*fd))
*(eps1*pow(e,(eps1*fd))-eps2*pow(e,(eps2*fd))));
//    gama=gamaDum-((tpInlet+b*(tg)-ts)*eps2*pow(e,(eps2*fd))+tg)/(eps1*pow(e,(eps1*fd))-
eps2*pow(e,(eps2*fd)));
//    gama=deltaT/fd-((tpInlet+b*(tg)-
ts)*eps2*pow(e,(eps2*fd))+tg)/(eps1*pow(e,(eps1*fd))-eps2*pow(e,(eps2*fd)));
//    gama=gamaDum+((-tpInlet-b*(tg)+ts)*eps1*pow(e,(eps1*fd))-
tg)/(eps1*pow(e,(eps1*fd))-
eps2*pow(e,(eps2*fd)))*eps2*pow(e,(eps2*fd))/eps1*pow(e,(eps1*fd))-
tg/eps1*pow(e,(eps1*fd));
    if (fail==0)
     delta1=(eps1*pow(e,(eps1*fd))-eps2*pow(e,(eps2*fd)));
    delta=((tpInlet+b*(tg)-ts)*eps1*pow(e,(eps1*fd))+tg-deltaT/fd/3.)/delta1;

 // original
 //   tp=k1*pow(e,(c1*j))+k2*pow(e,(c2*j))+tf-tg*a;           // K
 //   ta=k1*c3*pow(e,(c1*j))+k2*c4*pow(e,(c2*j))+tf;          // K

  // modified
    tfCalculation(j);
    tf=tfModified;
```

```
   // modified annulus temperature
    if (j==1)
    {
//      ta1=(1.+eps1*b)*gama*pow(e,(eps1*j))+(1.+eps2*b)*delta*pow(e,(eps2*j))+tg*j+ts-
deltaT/fd*j;
//      ta2=(1.+eps1*b)*gama*pow(e,(eps1*j))+(1.+eps2*b)*delta*pow(e,(eps2*j))+tg*j+ts-
deltaT;
      ta1=(1.+eps1*b)*gama*pow(e,(eps1*j))+(1.+eps2*b)*delta*pow(e,(eps2*j))+tg*j+ts;
      ta2=(1.+eps1*b)*gama*pow(e,(eps1*j))+(1.+eps2*b)*delta*pow(e,(eps2*j))+tg*j+ts;
      ta=(ta1+ta2)/2.;
    }
    else
    {
//      ta1=(1.+eps1*b)*gama*pow(e,(eps1*j))+(1.+eps2*b)*delta*pow(e,(eps2*j))+tg*j+ts-
deltaT/fd*j;
//      ta2=(1.+eps1*b)*gama*pow(e,(eps1*j))+(1.+eps2*b)*delta*pow(e,(eps2*j))+tg*j+ts-
deltaT;
      ta1=(1.+eps1*b)*gama*pow(e,(eps1*j))+(1.+eps2*b)*delta*pow(e,(eps2*j))+tg*j+ts;
      ta2=(1.+eps1*b)*gama*pow(e,(eps1*j))+(1.+eps2*b)*delta*pow(e,(eps2*j))+tg*j+ts;
      ta=(ta1+ta2)/2.;
    }

    if (ta<1.)
    {
      ta=10.;
      fail=1;
    }

   // modified pipe temperature

    if (j==1)
    {
      tp1=tpInlet;
//      tp2=gama*pow(e,(eps1*j))+delta*pow(e,(eps2*j))+tg*j-b*(deltaT/fd+tg)+ts;     //
modified due to Joule Thompson
      tp2=gama*pow(e,(eps1*j))+delta*pow(e,(eps2*j))+tg*j-b*tg+ts;
    }
    else
    {
//      tp1=gama*pow(e,(eps1*j))+delta*pow(e,(eps2*j))+tg*j-b*(deltaT/fd+tg)+ts;     //
modified due to Joule Thompson
//      tp2=gama*pow(e,(eps1*j))+delta*pow(e,(eps2*j))+tg*j-b*(deltaT/fd+tg)+ts;     //
modified due to Joule Thompson
      tp1=gama*pow(e,(eps1*j))+delta*pow(e,(eps2*j))+tg*j-b*tg+ts;
      tp2=gama*pow(e,(eps1*j))+delta*pow(e,(eps2*j))+tg*j-b*tg+ts;

    }
    tp=(tp1+tp2)/2.;

    if (tp<1.)
    {
      tp=10.;
      fail=1;
    }

    tpC1=tp;
    taC1=ta;
```

```
  // temperature calculation methodology - II
 /*
    a=(massFlowRate*fluidSpecificHeatPipe)/(2.*pi*rpi*uP);    // has a unit of meter
    b=(rb*uA)/(rpi*uP);    // dimensionless

    c1=(b/(2.*a))*(1.+pow((1.+4./b),0.5));       // has a unit of 1/meter
    c2=(b/(2.*a))*(1.-pow((1.+4./b),0.5));       // has a unit of 1/meter

    c3=1.+(b/2.)*(1.+pow((1.+4./b),0.5));        // dimensionless
    c4=1.+(b/2.)*(1.-pow((1.+4./b),0.5));        // dimensionless

   // unit has changed to F due to modifications conducted
    k2=(tg*a-(tpInlet-ts+tg*a)*pow(e,(c1*h))*(1.-c3))/(pow(e,(c2*h))*(1.-c4)-
pow(e,(c1*h))*(1.-c3));    // K
    k1=tpInlet-k2-ts+tg*a;    // K

  // original
 //   tp=k1*pow(e,(c1*j))+k2*pow(e,(c2*j))+tf-tg*a;           // K
 //   ta=k1*c3*pow(e,(c1*j))+k2*c4*pow(e,(c2*j))+tf;          // K

  // modified

    tfCalculation(j);
    tf=tfModified;

   // modified pipe temperature
    if (j==1)
    {
      tp1=k1*pow(e,(c1*j))+k2*pow(e,(c2*j))+tf-tg*a;           // K
      tp2=k1*pow(e,(c1*j))+k2*pow(e,(c2*j))+(temperatureWellbore[0][j]+tf)/2.-tg*a;
    }
    else
    {
      tp1=k1*pow(e,(c1*j))+k2*pow(e,(c2*j))+tf-tg*a;
      tp2=k1*pow(e,(c1*j))+k2*pow(e,(c2*j))+(temperatureWellbore[0][j]+tf)/2.-tg*a;
    }
    tp=(tp1+tp2)/2.;

   // modified annulus temperature
    if (j==1)
    {
      ta1=k1*c3*pow(e,(c1*j))+k2*c4*pow(e,(c2*j))+tf-delT[j];
      ta2=(k1*c3*pow(e,(c1*j))+k2*c4*pow(e,(c2*j))+(tf+temperatureWellbore[0][j])/2.)-
delT[j];
      ta=(ta1+ta2)/2.;
    }
    else
    {
      ta1=k1*c3*pow(e,(c1*j))+k2*c4*pow(e,(c2*j))+tf-delT[j];
      ta2=(k1*c3*pow(e,(c1*j))+k2*c4*pow(e,(c2*j))+(tf+temperatureWellbore[0][j])/2.)-
delT[j];
      ta=(ta1+ta2)/2.;
    }

    tpC2=tp;
    taC2=ta;
```

```
   // combining method - I and method - II
    temperatureInsidePipe[1][j]=(tpC1+tpC2)/2.;

    temperatureInsideAnnulus[1][j]=(taC1+taC2)/2.;
 */
  // should be removed if method-II is also used
    temperatureInsidePipe[1][j]=tpC1;
    temperatureInsideAnnulus[1][j]=taC1;

    parUa[j]=uA;
    parUp[j]=uP;
}


void bitPressure(double pipeT, double annT)
{
  double a,pA,k,tA;
//  double q;
  double flowRateF, flowRateW, flowRateT;
  int fd;

    flowRateF=massFlowRateFluid/fluidDensity;
    flowRateW=massFlowRateWater/waterDensity;
    flowRateT=flowRateF+flowRateW;

    k=cPovercV;
    fd=finalDepth;
    fluidPropertyDetermination(pipeT, pressureInsidePipe[0][fd]/144., fd);

    flowRateF=massFlowRateFluid/fluidDensity;
    flowRateW=massFlowRateWater/waterDensity;
    flowRateT=flowRateF+flowRateW;

    fluidPropertyMixture();
   // pipeT is the temperature at the bottom inside the pipe
   // annT is the termperature at the bottom inside the wellbore
   //   q=massFlowRate/fluidDensity;
    a=pi/4.*numberOfNozzles*pow(nozzleSize,2.);


bitPressureDrop=0.00008311*144.*fluidDensity/7.48*pow((flowRate/8.020833),2.)/(pow(0.95,2
.)*pow(a,2.));
    pA=pressureInsidePipe[0][fd]-bitPressureDrop;

    if (pA<0)
     pA=pressureInsideAnnulus[0][fd];

    fluidPropertyDetermination(annT, pA/144., fd);
    fluidPropertyMixture();

    tA=pipeT/pow((pressureInsidePipe[0][fd]/pressureInsideAnnulus[0][fd]),(k-1.)/k);

  //  deltaT=bitPressureDrop/144.*jouleThompsonCoefficient;
    deltaT=(pipeT-tA)*(1-flowRateW/flowRateT);

}
```

```
void pressureCalculation()
{
  int w, infD, fd;
  double t, p, tp;
  double pa, pp, dpf, dpfT, phyd, phydT;
  double nre, v, q, dhyd, a, ff;
  double mixF, mixI, mFR, mA;
  double solidCc;

    pa=backPressure;
    infD=influxDepth;
    fd=finalDepth;

    dpfT=0.;
    phydT=0.;

   // start from exit towards the bottom through the annulus
    for (w=1; w<=finalDepth; w++)
    {
      t=temperatureInsideAnnulus[1][w];
      p=pressureInsideAnnulus[0][w];
      fluidPropertyDetermination(t, p/144., w);
      fluidPropertyMixture();
      mA=plasticViscosity;

      mFR=massFlowRate;
     // influx contribution
      influxMassRate=prIndex*(formationPressureGradient*influxDepth-
pressureInsideAnnulus[0][infD])*influxDensity;

mixF=(massFlowRate/fluidDensity)/((massFlowRate/fluidDensity)+(influxMassRate/influxDensi
ty));

mixI=(influxMassRate/influxDensity)/((massFlowRate/fluidDensity)+(influxMassRate/influxDe
nsity));

     // solids contribution

solidCc=cuttingsConcentration[w]/100.+(massRateAbrasives/densityAbrasives)/((massFlowRate
/fluidDensity)+(influxMassRate/influxDensity)+(massRateAbrasives/densityAbrasives));
      if (solidCc >= 1.0)
       solidCc=0.99;

      if (w<=infD)
      {
        if ((formationPressureGradient*influxDepth)>pressureInsideAnnulus[0][infD])
        {
          t=t*mixF+temperatureFormation[infD]*mixI;
          fluidPropertyDetermination(t, p/144., w);
          fluidPropertyMixture();
          mA=mixF*plasticViscosity+mixI*plasticViscosityWater;
          mFR=mFR+influxMassRate;
        }
      }
```

```
     // hydrostatic pressure
      phyd=(fluidDensity*mixF+influxDensity*mixI)*(1-
solidCc)+(densityAbrasives+formationDensity)/2.*solidCc;
      hydrostaticAnnulus[w]=phyd+phydT;
      phydT=hydrostaticAnnulus[w];

     // frictional pressure
      phyd=fluidDensity*mixF+influxDensity*mixI;
      q=mFR/(phyd*60.);
      a=pi/4.*(pow(boreSize[w],2.)-pow(stringOD[w],2.));
      dhyd=boreSize[w]-stringOD[w];
      v=q/a;
      nre=phyd*v*0.816*dhyd*60./mA;

      if (nre<=2100)
       ff=16./nre;
      else
       ff=0.0791/pow(nre,0.25);

//      if (fluidPhase==1)
//       ff=0.56/(pow(dhyd,0.333));

      dpf=2*ff*phyd*pow(v,2.)/(dhyd*0.816)/(32.2*3600.);
      frictionAnnulus[w]=dpf+dpfT;
      dpfT=frictionAnnulus[w];

     // total pressure
      pa=pa+dpf+phyd;
      pressureInsideAnnulus[1][w]=pa;

      phaseInsideAnnulus[w]=fluidPhase;

    }


   // call bit pressure loss
    tp=temperatureInsidePipe[1][w-1];
    bitPressure(tp, t);

   // bit upstream pressure = pressure inside the pipe at the bottom
    bitUpstreamPressure=pressureInsideAnnulus[1][w-1]+bitPressureDrop;

    pp=bitUpstreamPressure;
    pressureInsidePipe[0][w-1]=pp;

    dpfT=0.;
    phydT=0.;

   // from bottom to surface through the pipe
    for (w=finalDepth; w>=1; w--)
    {
      t=temperatureInsidePipe[1][w];
      p=pressureInsidePipe[0][w];
      fluidPropertyDetermination(t, p/144., w);
      fluidPropertyMixture();

     // hydrostatic pressure
```

```
solidCc=(massRateAbrasives/densityAbrasives)/((massFlowRate/fluidDensity)+(massRateAbrasi
ves/densityAbrasives));
      if (solidCc >= 1.0)
       solidCc=0.99;

      phyd=fluidDensity*(1-solidCc)+densityAbrasives*solidCc;
      hydrostaticPipe[fd+1-w]=phyd+phydT;
      phydT=hydrostaticPipe[fd+1-w];

     // frictional pressure
      q=massFlowRate/(fluidDensity*60.);
      a=pi/4.*pow(stringID[w],2.);
      dhyd=stringID[w];
      v=q/a;
      nre=fluidDensity*v*dhyd*60./plasticViscosity;

      if (nre<=2100)
       ff=16./nre;
      else
       ff=0.0791/pow(nre,0.25);

//      if (fluidPhase==1)
//       ff=0.56/(pow(dhyd,0.333));

      dpf=2*ff*fluidDensity*pow(v,2.)/dhyd/(32.2*3600.);

      frictionPipe[w]=dpf+dpfT;
      dpfT=frictionPipe[w];
     // total pressure
      pp=pp+dpf-phyd;
      pressureInsidePipe[1][w]=pp;

      phaseInsidePipe[w]=fluidPhase;
    }

    for (w=1; w<=finalDepth; w++)
    {
      pressureInsidePipe[0][w]=pressureInsidePipe[1][w];
      pressureInsideAnnulus[0][w]=pressureInsideAnnulus[1][w];
    }


}


void initialize()
{

  int i, fd;

    fd=finalDepth;

    for (i=1; i<=fd; i++)
    {
     // initial temperature set (formation temperature)
      temperatureWellbore[0][i]=surfaceTemperature;
      temperatureInsidePipe[0][i]=surfaceTemperature+(temperatureGradient)*i;
```

```
      temperatureInsideAnnulus[0][i]=surfaceTemperature;
      temperatureFormation[i]=surfaceTemperature+(temperatureGradient)*i;

      fluidPropertyDetermination(temperatureInsideAnnulus[0][i],
pressureInsideAnnulus[0][i]/144., i);

      pressureInsideAnnulus[0][i]=backPressure+fluidDensity*i;

      phaseInsideAnnulus[i]=0;
      phaseInsidePipe[i]=0;

      cuttingsConcentration[i]=0;
    }

    temperatureInsidePipe[0][1]=fluidInletTemperature;
    tBottomhole=temperatureInsideAnnulus[0][fd];

    for (i=1; i<=fd; i++)
    {
     // initial guess (if you find a better dPf estimation (empirical), use it and modify
the equation
      pressureInsidePipe[0][i]=pressureInsideAnnulus[0][fd];
    }

    fluidPropertyDetermination(temperatureInsidePipe[0][1],
pressureInsidePipe[0][1]/144., 1);
    bitPressureDrop=0.;
    massFlowRateFluid=fluidDensity*flowRateFluid;
    massFlowRateWater=waterDensity*flowRateWater;
    massFlowRate=massFlowRateFluid+massFlowRateWater;

}


void geometryDescription()
{
  int i;

  for (i=1; i<=finalDepth; i++)
  {
     if (i<=casingSettingDepth)
     {
       boreSize[i]=casingID;
       boreSizeOld[i]=holeSize;
       casOD[i]=casingOD;
       casID[i]=casingID;
     }
     else
     {
       boreSize[i]=bitDiameter;
       boreSizeOld[i]=bitDiameter;
       casOD[i]=bitDiameter;
       casID[i]=bitDiameter;
     }

     if (i>(finalDepth-drillCollarLength))
     {
       stringOD[i]=drillCollarOD;
```

```
        stringID[i]=drillCollarID;
      }
      else
      {
        stringOD[i]=drillPipeOD;
        stringID[i]=drillPipeID;
      }

  }


}



void convergenceCheck()
{
  int errC;

    maxErr=0;
    err=0;
    errDepth=0;

    for (errC=1; errC<=finalDepth; errC++)
    {
      err=temperatureInsidePipe[1][errC]-temperatureInsidePipe[0][errC];
      if (err<0)
       err=-err;
      if (err>maxErr)
      {
        maxErr=err;
        errDepth=errC;
      }
      temperatureInsidePipe[0][errC]=temperatureInsidePipe[1][errC];

      err=temperatureInsideAnnulus[1][errC]-temperatureInsideAnnulus[0][errC];
      if (err<0)
       err=-err;
      if (err>maxErr)
      {
        maxErr=err;
        errDepth=errC;
      }
      temperatureInsideAnnulus[0][errC]=temperatureInsideAnnulus[1][errC];

      err=temperatureWellbore[1][errC]-temperatureWellbore[0][errC];
      if (err<0)
       err=-err;
      if (err>maxErr)
      {
        maxErr=err;
        errDepth=errC;
      }
      temperatureWellbore[0][errC]=temperatureWellbore[1][errC];
    }
}
```

```
// cuttings transport verification
void cuttingsTransportCheck()
{
  double dc, a, b, denC, denF, vsOld, vs, me, cc;
  double nRep, vf, errSlip;
  double aw, ab;
  int i;
  const g=32.198;


    dc=cuttingsSize;
    denC=formationDensity;

    for (i=1; i<=finalDepth; i++)
    {

      vs=1000.;
      errSlip=1000.;

      fluidPropertyDetermination(temperatureInsideAnnulus[1][i],
pressureInsideAnnulus[1][i]/144., i);
      fluidPropertyMixture();

      denF=fluidDensity;

      aw=(pi/4)*(pow(casID[i],2.)-pow(stringOD[i],2.));
      ab=(pi/4)*pow(casID[i],2.);
      vf=massFlowRate/(denF*aw);

      do
      {
        vsOld=vs;
        me=plasticViscosity+yieldPoint*dc/vf;
        nRep=dc*denF*vsOld/me;

        if (nRep<=1.)
        {
          a=40.;
          b=-1.;
        }
        if ((nRep>1.) && (nRep<1000.))
        {
          a=22.;
          b=-0.5;
        }
        if (nRep>=1000.)
        {
          a=1.5;
          b=0.;
        }

        vs=pow(((dc*g*pow(3600.,2.)/a)*(denC-denF)/denF*pow(nRep,-b)),0.5);
        errSlip=vs-vsOld;
        if (errSlip<0)
         errSlip=-errSlip;
      }
      while (errSlip>0.0001);
```

```
    cc=(ROP*ab)/(aw*(vf-vs));

    cuttingsConcentration[i]=cc*100.;
    transportRatio=1.-vs/vf;
    }
}



// main program
#pragma argsused
int main(int argc, char* argv[])
{

 double massOld, massErr;
 int tt, dd, counter, errCounter, cntr;
 char phasePipe[10], phaseAnnulus[10];

  fail=0;
  intro();

  do
  {

    choice=0;

    intro1();

 //   cin >> choice;

    choice=1;
    switch (choice)
    {
      case 0: // Exit
      {
        exit('0');
        break;
      }

      case 1:  // Continue
      {
  //      clrscr();

        inputPage();

        unitConversion();
     //  main

        finalCirculationTime=5.;  // define this properly later
        circulationTime=0;
        dcirculationTime=5.;

        geometryDescription();
  //      printf("Calculation in progress ");

        ofstream outfileDum ("out.lvm");
        outfileDum.close();
```

```
        ofstream outfile ("out.lvm", ios::app);

       // initiate the file
        outfile << "t(hr) " << "D(ft) " << "Tann(F) " << "Tpip(F) " << "Twb.(F) " <<
"Tfor(F) " << "Pa(psi) " << "Pp(psi) " << "PfrA(psi) " << "PfrP(psi) " << "PhydA(psi) "
<< "PhydP(psi) " << "phaseA " << "phaseP" << "Cc " << "Rt " << endl;
 //       outfile << "-------------------------------------------------------"<< endl;

       // loop for time
        while (circulationTime<finalCirculationTime)
        {
 //         printf(".");
          initialize();

          printErr=0;
          errCounter=0;
          circulationTime=circulationTime+dcirculationTime;

          cntr=0;
       // loop for convergence
          do
          {
            if (fail==1)
             goto unstableCase;
            cntr=cntr+1;
            massOld=massFlowRate;
            errCounter=errCounter+1;

           // depth loop
            for (dd=1; dd<=finalDepth; dd++)
            {
              temperatureCalculation(dd, temperatureWellbore[0][dd]);
            }
            pressureCalculation();

            temperatureUpdate();

            convergenceCheck();

            if (errCounter>50)
            {
              printErr=1;
              printMaxErr=maxErr;
              maxErr=0.01;
            }

            fluidPropertyDetermination(temperatureInsidePipe[0][1],
pressureInsidePipe[0][1]/144., 1);
            massFlowRateFluid=fluidDensity*flowRateFluid;
            massFlowRateWater=waterDensity*flowRateWater;
            massFlowRate=massFlowRateFluid+massFlowRateWater;


         // to make sure mass rate is stable between two iterations
            massErr=massFlowRate-massOld;
            if (massErr<0)
             massErr=-massErr;
```

```
            if (cntr>200)
             massErr=0.9;
          }
          while ((maxErr>=0.05) || (massErr>=1.));

          cuttingsTransportCheck();

         // writing information into a file
  //        circulationTime2=0;
  //        while (circulationTime2<finalCirculationTime)
  //        {
  //          circulationTime2=circulationTime2+dcirculationTime;

            for (counter=1; counter<=finalDepth; counter++)
            {
              if (phaseInsideAnnulus[counter]==0)
               strcpy(phaseAnnulus, "liquid");
              if (phaseInsideAnnulus[counter]==1)
               strcpy(phaseAnnulus, "gas");
              if (phaseInsideAnnulus[counter]==2)
               strcpy(phaseAnnulus, "supercrt");
              if (phaseInsidePipe[counter]==0)
               strcpy(phasePipe, "liquid");
              if (phaseInsidePipe[counter]==1)
               strcpy(phasePipe, "gas");
              if (phaseInsidePipe[counter]==2)
               strcpy(phasePipe, "supercrt");
  /*
         char op1[10];
         char op2[10];
         char op3[10];
         char op4[10];
         char op5[10];
         char op6[10];
         char op7[10];
         char op8[10];
         char op9[12];
         char op10[12];

         sprintf(op1, "%4d ", circulationTime);
         sprintf(op2, "%5d ", counter);
         sprintf(op3, "%5d ", temperatureInsideAnnulus[1][counter]);
         sprintf(op4, "%5d ", temperatureInsidePipe[1][counter]);
         sprintf(op5, "%5d ", temperatureWellbore[1][counter]);
         sprintf(op6, "%5d " ,temperatureFormation[counter]);
         sprintf(op7, "%7d ", pressureInsideAnnulus[1][counter]/144.);
         sprintf(op8, "%7d ", pressureInsidePipe[1][counter]/144.);
         sprintf(op9, "%9c ", phaseAnnulus);
         sprintf(op10, "%9c ", phasePipe);

              outfile << op1 << op2 << op3 << op4 << op5 << op6 << op7 << op8 << op9 <<
op10 << endl;
   */
              outfile << circulationTime << " " << counter << " " <<
temperatureInsideAnnulus[1][counter] << " " << temperatureInsidePipe[1][counter] << " "
<< temperatureWellbore[1][counter]
```

```
               << " " << temperatureFormation[counter] << " " <<
pressureInsideAnnulus[0][counter]/144. << " " << pressureInsidePipe[0][counter]/144. << "
" << frictionAnnulus[counter]/144. << " " << frictionPipe[counter]/144. << " " <<
               hydrostaticAnnulus[counter]/144. << " " << hydrostaticPipe[counter]/144.
<< " " << phaseAnnulus << " " << phasePipe << " " << cuttingsConcentration[counter] << "
" << transportRatio << " " << endl;

            }
   //       }
         }

   //     clrscr();

         outfile.close();
         outputPage();


   //     clearScreen();
         break;
      }


      default:  // continue
      {
         choice=1;

 //       clrscr();

         inputPage();
     // main

         outputPage();

 //       clearScreen();
         break;
      }
    }

  unstableCase:
   if (fail == 1)
   {
      clrscr();
  //    printf("Stable solution is not possible with the current input data.\n");
  //    printf("\n");
  //   printf("Either 'freezing' or 'physically impossible' situation.\n");
  //    printf("\n");
  //    printf("Suggestions:.\n");
  //    printf("  - Change nozzle size\n");
  //    printf("  - Change backpressure\n");
  //    printf("  - Change flow rate\n");
  //    printf("  - Add water into the system\n");
  //    printf("\n");
  //    printf("Please modify the input information.\n");
  //    printf("\n");
  //    printf("(Press any key to continue ...)\n");
  //    getch();
  //    clrscr();
      ofstream outfileDum ("out.lvm");
```

```
      outfileDum.close();
    }

  choice=0;
  }
  while (choice != 0);

    return 0;
}
//-----------------------------------------------------------------------------
```