«main.m» (Mecanistic Model)

```
% Standalone Network Solver for Short-Term prodcution optimization

% All Comments:
% 1. All properties may be included into structures (prop, rates, wells
etc.) for a cleaner look.



clear;
clear global;
clc;
%% Specify global variables and add folders to path

restoredefaultpath;
addpath(genpath('../Standalone_Network_Solver_Mec'))
format short g;
clc;

global dens_o_sc dens_w_sc gamma_o gamma_g_0 gamma_gl_0 visc_o_sc
visc_g_sc_0 pSep pBP_0 h_tubing l_tubing ...
    h_flowline l_flowline h_riser l_riser rough M_g p_sc T_sc g R wc GOR_0
nWells API_o API_w M_air qgl_sc_max...
    inch bara day qo_sc_min l_PLEM_WH h_PLEM_WH nManifold wells_in_manifold
ID_t ID_f ID_m ...
    Tbnoil Tbngas h_oil h_gas k_steel k_sand thickness_t thickness_m
thickness_f rlm_t rlm_m rlm_f ro_t ro_m ro_f ...
    h_sea k_ins ro_ins_m ro_ins_f T_sea step_length

%% Conversion factors

inch = 0.0254; % [m]
bara = 10^5;   % [pa]
day = 24*3600; % [s]



%% Import production profiles and pRes over time.
% Self generated or from simulator (Eclipse/MRST/Remso) using some model.



alpha = 1.3; %Parallel
alpha = alpha';
for i = 1:length(alpha)
    qo_sc(i, :) = [1200]*alpha(i)/day;
    qo_target(i, 1) =  sum(qo_sc(i, :))*day;
    pR(i, 1) = 190;   % All pressures in [Bar]
    TR_C = 100;
    TR(i, 1) = TR_C+273.15;
end

wc = [0.3];



% You can also choose to define GOR_0, and from there receive the pBP_0
GOR_0 = 500;   % Sm3/sm3, The "0"-values, eg. GOR_0 is value at reservoir
condition.
```

```
qw_sc = zeros(size(qo_sc));
qg_sc = zeros(size(qo_sc));
qgl_sc = zeros(size(qo_sc));
dp_choke = zeros(size(qo_sc));

for k =1:length(pR)
qw_sc(k, :) = qo_sc(k, :).*wc;
qg_sc(k, :) = qo_sc(k, :).*GOR_0;
end

qgl_sc_max = 50000/day;
qo_sc_min = 0.1/day;

%% Specify reservoir properties and constants

pSep = 7;
p_sc = 1.01325;
T_sc = 15.56+273.15;

g = 9.81; % [m/s^2]
R = 8314;  % [J/Kmol]



% As long as rates, pRes and time are given, the rest of the program works.
%% Specify fluid properties (Gas without GL)

% At SC
dens_o_sc = 780; % [kg/Sm3]
dens_w_sc = XSteam('rho_pT', p_sc, T_sc-273.15);
M_g = 19;  % [kg/kg-mole]
M_gl = 25;
M_air = 28.97;
gamma_g_0 = M_g/M_air;
gamma_gl_0 = M_gl/M_air;
gamma_o = dens_o_sc/dens_w_sc;
gamma_w = dens_w_sc/dens_w_sc;
API_o = 141.5/gamma_o-131.5;
API_w = 141.5/gamma_w-131.5;

pBP_0 = bubble_point(GOR_0, TR(k), gamma_g_0);

% At SC, "Dead o/w/g". Should depend upon sep. Conditions?
visc_o_sc = dead_visc_oil(T_sc, API_o);   % [cp]
%visc_w_sc = dead_visc_oil(T_sc, API_w);   % [cp]

visc_g_sc_0 = dead_visc_gas(gamma_g_0);

% Boiling points of the gas and liquid mixture:
Tbnoil = 851;   % Rankine
Tbngas = 196.47; % Rankine

% Convection heat transfer coefficients for oil and gas [W/m2 K]:
h_oil = 50;
h_gas = 11;

% Thermal conductivity of Steel and Sandstone
```

```
k_steel = 43; % (Stainless = 17. With 1% carbon, it's 43)
k_sand = 1.7;

%% Specify network properties
% May be tweeked for multiple wells, of course.

nWells = length(wc);
wellName =zeros(nWells, 1);
for i=1:nWells
wellName(i, 1) = i;
end

h_tubing = [2000]; % TVD [m]
l_tubing = [2000]; % Measured Length [m]
wells_in_manifold = [1]; % Number of wells in [first, second, ..., n.th]
manifold.
l_PLEM_WH = [800]; % Distance from PLEM to manifold 1 and 2. 2 Wells for
each manifold.
nManifold = length(l_PLEM_WH);
h_PLEM_WH = [30];
h_flowline = 400;
l_flowline = 3000;
h_riser = 100;
l_riser = 100;


% for i = 1:nManifold
%     temp_10=0;
%     for y = 1:i
%         temp_9 = temp_10;
%         temp_10 = temp_10+wells_in_manifold(y);
%     end
%     TVD(temp_9+1:temp_10) = h_tubing(temp_9+1:temp_10) + h_PLEM_WH(i) +
h_flowline + h_riser;
%
% end


ID_t = 5*inch; % Tubing ID [inch], for all pipes
ID_m = (wells_in_manifold.^0.5)*ID_t;
ID_f = ID_t*sqrt(nWells);

% Pipes thickness:
thickness_t = 0.5*inch; % [m]
thickness_m = 1*inch; % [m]
thickness_f = 1.15*inch; % [m]

% Outer radius [m]:
ro_t = ID_t/2 + thickness_t;
ro_m = ID_m/2 + thickness_m;
ro_f = ID_f/2 + thickness_f;

rlm_t = thickness_t/( log(ro_t) - log(ID_t/2));
rlm_m = thickness_m./( log(ro_m) - log(ID_m/2));
rlm_f = thickness_f/( log(ro_f) - log(ID_f/2));

% HTC in the sea:
% Outer radius of insulation [m]:
ro_ins_m = ro_m + 0.033;
```

```
ro_ins_f = ro_f + 0.033;
% Conductivity of insulation [W/m K]:
k_ins = 0.22;
% Sea convection coefficient [W/m2 K]:
h_sea = 200;
T_sea = 6+273.15;

rough = 2.54*10^-5; % Roughness For all pipes, not needed for Beggs&Brill
model [m]
%% Flow in pipes, DP calculations
time=[];
% Choose solver:
choice = menu('Choose an optimizer','Sequential Approach','Parallel
Approach');


% For running simulations with multiple, specified pR and rates.


for k=1:length (pR)
    disp(alpha(k));
tic;
if sum(qo_sc(k, :)  )+sum(qw_sc(k, :)   )==0  % Gas production only, use
compressors
        res = gasProd(qg_sc(k, :), pR(k));
else  % Multiphase flow, use GL or ESP
    if choice == 1
        [pTubing, pPLEM_WH, pRiser_Flow, p, T, qo_sc(k, :), qgl_sc(k, :),
dp_choke(k, :)] = GL_sequential_NR(qo_sc(k, :), pR(k), TR(k));
    elseif choice == 2
        [pTubing, pPLEM_WH, pRiser_Flow, p, qo_sc(k, :), qg_sc(k, :),
qw_sc(k, :), qgl_sc(k, :), dp_choke(k, :)] = GL_parallel_NR(qo_sc(k, :),
qw_sc(k, :), qg_sc(k, :), pR(k));
    end
end
time(k, 1) = toc;
qo_sum(k, 1) = sum(qo_sc(k, :));
GL_sum(k, 1) = sum(qgl_sc(k, :));

% fprintf('Simulation runtime [s]: %.2f. \nField Oil rate [Sm3/D]: %.2f
\nField Gas Lift rate [Sm3/D]: %.2f \n\n', time(k, 1), qo_sum(k, 1),
GL_sum(k, 1));
% %% Plot of results:
%
% % Flowline and riser:
% pRiser_Flow_plot = [];
%     for j = length(pRiser_Flow(:, 1)):-1:1
%         if pRiser_Flow(j, 1)>0
%             pRiser_Flow_plot = [pRiser_Flow_plot, pRiser_Flow(j, 1)];   %
Pressure values from Sep. to Bottom.
%         end
%     end
% lRiser_Flow_plot = (0:(length(pRiser_Flow_plot)-1))*step_length;
%
%
% figure(k)
% hold on;
% cmap = hsv(nWells);
%
% for i = 1:nWells
```

```
%
%     for z = 1:nManifold
%         temp_10=0;
%         for y = 1:z
%             temp_9 = temp_10;
%             temp_10 = temp_10+wells_in_manifold(y);
%         end
%         if i>temp_9 && i<=temp_10
%             % PLEM-WH:
%             pPLEM_WH_plot = [];
%             for j = length(pPLEM_WH(:, z)):-1:1
%                 if pPLEM_WH(j, z)>0
%                     pPLEM_WH_plot = [pPLEM_WH_plot, pPLEM_WH(j, z)];   %
Pressure values from Sep. to Bottom.
%                 end
%             end
%             lPLEM_WH_plot =
lRiser_Flow_plot(end):step_length:(lRiser_Flow_plot(end)+(length(pPLEM_WH_p
lot)-1)*step_length);
%             break;
%         end
%
%     end
%
%     % Tubing:
%     pTubing_plot = [];
%     for j = length(pTubing(:, i)):-1:1
%         if pTubing(j, i)>0
%             pTubing_plot = [pTubing_plot, pTubing(j, i)];   % Pressure
values from Sep. to Bottom.
%         end
%     end
%     lTubing_plot =
lPLEM_WH_plot(end):step_length:(lPLEM_WH_plot(end)+(length(pTubing_plot)-
1)*step_length);
%     % Choke:
%     pChoke_plot = [pPLEM_WH_plot(end) pTubing_plot(1)];
%     lChoke_plot = [lPLEM_WH_plot(end) lPLEM_WH_plot(end)];
%     % Total plot:
%     lPlot = [lRiser_Flow_plot, lPLEM_WH_plot, lChoke_plot, lTubing_plot];
%     pPlot = [pRiser_Flow_plot, pPLEM_WH_plot, pChoke_plot, pTubing_plot];
%
%     plot( lPlot, pPlot, 'Color', cmap(i, :) );
%
%     %str = num2str(i);
%     legendInfo{i} = ['Well = ' num2str(i)];
% end
% legend(legendInfo, 'Location', 'NorthWest');
% title(['Pressure along pipe at time ', num2str(k)]);
% xlabel('Travel distance from Sep. [m]');
% ylabel('Pressure [bar]');
% hold off;

end
```

«main.m» (Compositional Model)

```
% Standalone Network Solver for Short-Term prodcution optimization

% All Comments:
% 1. All properties may be included into structures (prop, rates, wells
etc.) for a cleaner look.
% 4. Current example data are mostly from Exercise 5, TPG4230.


clear;
clear global;
clc;
%% Specify global variables and add folders to path

%restoredefaultpath;
addpath(genpath('../Standalone_Network_Solver_Comp'))
format short g;
clc;

global dens_o_sc dens_w_sc gamma_o gamma_g_0 gamma_gl_0 visc_o_sc
visc_g_sc_0 pSep pBP_0 h_tubing l_tubing ...
    h_flowline l_flowline h_riser l_riser rough M_g p_sc T_sc g R wc GOR_0
nWells API_o API_w M_air qgl_sc_max...
    inch bara day qo_sc_min l_PLEM_WH h_PLEM_WH nManifold wells_in_manifold
ID_t ID_f ID_m ...
    Tbnoil Tbngas h_oil h_gas k_steel k_sand thickness_t thickness_m
thickness_f rlm_t rlm_m rlm_f ro_t ro_m ro_f ...
    h_sea k_ins ro_ins_m ro_ins_f T_sea MolarFlowGL_max Fv_sc Ml_sc Mg_sc
dens_g_sc MolarFlowTot_min

%% Conversion factors

inch = 0.0254; % [m]
bara = 10^5;   % [pa]
day = 24*3600; % [s]

time=[];

%% Import production profiles and pRes over time.
% Self generated or from simulator (Eclipse/MRST/Remso) using some model.



alpha = 2.1:0.1:5; %Parallel
alpha = alpha';
for i = 1:length(alpha)
    qo_sc(i, :) = [1000]*alpha(i)/day;
    qo_target(i, 1) =  sum(qo_sc(i, :))*day;
    pR(i, 1) = 190;   % All pressures in [Bar]

end

wc = [0.3];
```

```
% You can also choose to define GOR_0, and from there receive the pBP_0
GOR_0 = 100;   % Sm3/sm3, The "0"-values, eg. GOR_0 is value at reservoir
condition.

qw_sc = zeros(size(qo_sc));
qg_sc = zeros(size(qo_sc));
qgl_sc = zeros(size(qo_sc));
dp_choke = zeros(size(qo_sc));
for k=1:length (pR)
% for k =1:length(pR)
% qw_sc(k, :) = qo_sc(k, :).*wc;
% qg_sc(k, :) = qo_sc(k, :).*GOR_0;
% end

qgl_sc_max = 50000/day;
%qgl_sc_max =0;
MolarFlowTot_min = 0.1/day;


%% Specify reservoir properties and constants

pSep = 7;
p_sc = 1.1;
T_sc = 15.56+273.15;
TR_C = 100;
TR = TR_C+273.15;
g = 9.81; % [m/s^2]
R = 8314;  % [J/Kmol]

[MolarFlowTot, MolarFlowGL_max, dens_o_sc, dens_g_sc, dens_w_sc, y_GL,
Fv_sc, Ml_sc, Mg_sc] = getInitPVTproperties(T_sc, p_sc, qo_sc(k),
qgl_sc_max);
y_GL = zeros(length(y_GL), 1);
y_GL (3) = 1;

% As long as rates, pRes and time are given, the rest of the program works.
%% Specify fluid properties (Gas without GL)

% At SC
% dens_o_sc = 780; % [kg/Sm3]
% dens_w_sc = XSteam('rho_pT', p_sc, T_sc-273.15);
% M_g = 19;  % [kg/kg-mole]
% M_gl = 25;
% M_air = 28.97;
% gamma_g_0 = M_g/M_air;
% gamma_gl_0 = M_gl/M_air;
 gamma_o = dens_o_sc/dens_w_sc;
 gamma_w = dens_w_sc/dens_w_sc;
% API_o = 141.5/gamma_o-131.5;
% API_w = 141.5/gamma_w-131.5;
%
% pBP_0 = bubble_point(GOR_0, TR, gamma_g_0);
%
% % At SC, "Dead o/w/g". Should depend upon sep. Conditions?
% visc_o_sc = dead_visc_oil(T_sc, API_o);   % [cp]
% %visc_w_sc = dead_visc_oil(T_sc, API_w);   % [cp]
%
% visc_g_sc_0 = dead_visc_gas(gamma_g_0);

% Boiling points of the gas and liquid mixture:
```

```
% Tbnoil = 851;   % Rankine
% Tbngas = 196.47; % Rankine

% Convection heat transfer coefficients for oil and gas [W/m2 K]:
h_oil = 50;
h_gas = 11;

% Thermal conductivity of Steel and Sandstone
k_steel = 43; % (Stainless = 17. With 1% carbon, it's 43)
k_sand = 1.7;

%% Specify network properties
% May be tweeked for multiple wells, of course.

nWells = length(wc);
wellName =zeros(nWells, 1);
for i=1:nWells
wellName(i, 1) = i;
end

h_tubing = [2000]; % TVD [m]
l_tubing = [2000]; % Measured Length [m]
wells_in_manifold = [1]; % Number of wells in [first, second, ..., n.th]
manifold.
l_PLEM_WH = [100]; % Distance from PLEM to manifold 1 and 2. 2 Wells for
each manifold.
nManifold = length(l_PLEM_WH);
h_PLEM_WH = [30];
h_flowline = 500;
l_flowline = 2000;
h_riser = 100;
l_riser = 100;

% h_tubing = [2000]; % TVD [m]
% l_tubing = [2000]; % Measured Length [m]
% wells_in_manifold = [1]; % Number of wells in [first, second, ..., n.th]
manifold.
% l_PLEM_WH = [0]; % Distance from PLEM to manifold 1 and 2. 2 Wells for
each manifold.
% nManifold = length(l_PLEM_WH);
% h_PLEM_WH = [0];
% h_flowline = 0;
% l_flowline = 2000;
% h_riser = 500;
% l_riser = 500;


% for i = 1:nManifold
%     temp_10=0;
%     for y = 1:i
%         temp_9 = temp_10;
%         temp_10 = temp_10+wells_in_manifold(y);
%     end
%     TVD(temp_9+1:temp_10) = h_tubing(temp_9+1:temp_10) + h_PLEM_WH(i) +
h_flowline + h_riser;
%
% end
```

```
ID_t = 5*inch; % Tubing ID [inch], for all pipes
ID_m = (wells_in_manifold.^0.5)*ID_t;
ID_f = ID_t*sqrt(nWells);

% Pipes thickness:
thickness_t = 0.5*inch; % [m]
thickness_m = 1*inch; % [m]
thickness_f = 1.15*inch; % [m]

% Outer radius [m]:
ro_t = ID_t/2 + thickness_t;
ro_m = ID_m/2 + thickness_m;
ro_f = ID_f/2 + thickness_f;

rlm_t = thickness_t/( log(ro_t) - log(ID_t/2));
rlm_m = thickness_m./( log(ro_m) - log(ID_m/2));
rlm_f = thickness_f/( log(ro_f) - log(ID_f/2));

% HTC in the sea:
% Outer radius of insulation [m]:
ro_ins_m = ro_m + 0.033;
ro_ins_f = ro_f + 0.033;
% Conductivity of insulation [W/m K]:
k_ins = 0.22;
% Sea convection coefficient [W/m2 K]:
h_sea = 200;
T_sea = 6+273.15;

rough = 2.54*10^-5; % Roughness For all pipes, not needed for Beggs&Brill
model [m]
%% Flow in pipes, DP calculations
% Choose solver:

    disp(alpha(k));
tic;

        [pTubing, pPLEM_WH, pRiser_Flow, p, T, qo_sc(k, :), qgl_sc(k, :),
dp_choke(k, :)] = GL_sequential_NR(MolarFlowTot, y_GL, pR(k), TR);


time(k, 1) = toc;
qo_sum(k, 1) = sum(qo_sc(k, :));
GL_sum(k, 1) = sum(qgl_sc(k, :));


end
```

«getInitPVTproperties.m»

```
function [MolarFlowTot, MolarFlowGL, OilDensity, GasDensity, WaterDensity,
y, Fv, Ml, Mv]=getInitPVTproperties(T, P, qo_sc, qgl_sc)


NRerror = 1e-2;          % Newton-Rhapson error tolerance
e = 1e-2;                % Fugacity convergence tolerance

% Pl = 37;                 % Lower pressure limit [bar]
% Pu = 100;                % Upper pressure limit [bar]
% dP = 1;                  % Delta pressure [bar]


% T and p at Surface Conditions
% qo_sc is the suggested surface oil rate.
% qgl_sc is the *maximum* GL rate at surface


frr=1;                                  % Avtivate the fugacity while-loop

R = 8.314E-5;                             % Gas constant
FluidProp = xlsread ( 'FluidProp' );      % Fluid properties
Pci = FluidProp(:,4);                     % Critical pressure [bar]
Tci = FluidProp(:,5);                     % Critical temperature [K]
%Vci = FluidProp(:,6);                     % Critical volume [m3/kmol]
kij = xlsread ( 'PR-EOS-BIPS' );          % BIP's
kij= kij + kij';
z = FluidProp(:,1);                       % Mixture composition
Mi = FluidProp(:,2);                      % Molar weight kg/kmol
AF = FluidProp(:,8);                      % Acentric factor
VTP = FluidProp(:,11);                    % PR Volume translation parameters




%% K Value estimate; Wilson equation

Pri = P ./ Pci;                          % Reduced pressure
Tri = T ./ Tci;                          % Reduced temperature



K = exp(5.37 * (1 + AF) .* (1 - 1 ./ Tri)) ./ Pri ;   % Phase equilibrium
factor

m = zeros(length(z), 1);
for j = 1 : length(z)

    if AF(j) < 0.4

        m(j) = 0.37464 + 1.54226 * AF(j) - 0.26992 * AF(j)^2;

    else
```

```
        m(j) = 0.3796 + 1.485 * AF(j) - 0.1644 * AF(j)^2 + 0.01667 *
AF(j)^3;

    end
end

alfa = (1 + m .* (1 - sqrt(Tri))).^2;

a = 0.45724 * R^2 * Tci.^2 ./ Pci .* alfa;

b = 0.07780 * R * Tci ./ Pci;

A = a * P / (R * T)^2;

B = b * P / (R * T);

%% Fv calculation; Rachford-Rice

while frr>e      %Fugacity tolerance constraint


Fvmin = 1 / (1 - max(K));
Fvmax = 1 / (1 - min(K));


%syms Fv        % Matlab numerical solver
%Fv=double(vpasolve(sum(z .* (K - 1) ./ (1 + Fv(1) * (K - 1))),Fv,[Fvmin
Fvmax]));


% Newton-Raphson method
Fv(1) = (Fvmin+Fvmax)/2;            % Initial guess

hFv = sum(z .* (K - 1) ./ (1 + Fv(1) * (K - 1)));

dhFv = - sum(z .* (K - 1).^2 ./ (Fv(1) * (K - 1) + 1).^2);

Fv(2) = Fv(1) - hFv / dhFv;

while abs(Fv(1) - Fv(2)) > NRerror     % Error tolerance

    Fv(1) = Fv (2);

    hFv = sum(z .* (K - 1) ./ (1 + Fv(1) * (K - 1)));

    dhFv = - sum(z .* (K - 1).^2 ./ (Fv(1) * (K - 1) + 1).^2);

    Fv(2) = Fv(1) - hFv / dhFv;

end


if Fv(2) > Fvmin && Fv(2) < Fvmax
    Fv = Fv(2);
end
```

```
% Liquid and vapor phase composition

x = z ./ (Fv .* (K - 1) + 1);

y = K .* x;


%% A / B parameters

Aij = (1-kij) .* sqrt(A * A');

AL = x' * Aij * x;

AV = y' * Aij * y;

BV = y' * B;

BL = x' * B;

syms Zl Zv

Zl = real(double(solve(Zl^3 - (1 - BL)*Zl^2 + (AL -3*BL^2-2*BL)*Zl-(AL*BL-
BL^2-BL^3))));

Zl = Zl(1);


Zv = real(double(solve(Zv^3 - (1 - BV)*Zv^2 + (AV -3*BV^2-2*BV)*Zv-(AV*BV-
BV^2-BV^3))));

Zv = Zv(1);


%% Fugacity

% Liquid fugacity
fl = exp(B/BL*(Zl-1)-log(Zl-BL)+AL/(2*sqrt(2)*BL)*(B/BL-
2/AL*Aij*x)*log((Zl+(1+sqrt(2))*BL)/(Zl+(1-sqrt(2))*BL))+log(x*P));

% Vapor fugacity
fv = exp(B/BV*(Zv-1)-log(Zv-BV)+AV/(2*sqrt(2)*BV)*(B/BV-
2/AV*Aij*y)*log((Zv+(1+sqrt(2))*BV)/(Zv+(1-sqrt(2))*BV))+log(y*P));

% Fugacity ratio
fr=fl./fv;

% K value update
K=K.*fr;

% Convergence test
frr=sum((1-fr).^2);
end



% Fv<0 => Single phase, liquid
```

```
if Fv<0

    Fv = 0;
    x=z;
    y=zeros(length(z),1);


% Fv>1 => Single phase, vapor

elseif Fv>1;

    Fv = 1;
    y=z;
    x=zeros(length(z),1);

end

%Save Fv for every pressure calculation in a vector
%Fvv=Fv;

%% Vapor properties


Mv = Mi'*y;                                         % Vapor phase molar weight

%GasDensityEOS = P * Mv*1E-3 / (Zv * R * T);    % EOS calculated vapor
density
vvEOS = Zv * R * T* 1E+3/P;                         % EOS calculated vapor
molar volume

c = b*1E3 .* VTP;                                      % Volume translation
parameter

vvCOR = vvEOS - sum(c.*y);                      % Corrected vapor molar
volume
GasDensity = Mv / vvCOR;                % Corrected vapor density



%% Liquid properties

Ml = Mi'*x;                                         % Liquid phase molar
weight

vlEOS = Zl * R * T* 1E+3/P;                         % Liquid phase molar
volume

c = b*1E3 .* VTP;                                      % Volume translation
parameter

vlCOR = vlEOS - sum(c.*x);                      % Corrected liquid phase
molar volume


OilDensity = Ml / vlCOR;                % Corrected liquid density
```

```
WaterDensity = XSteam('rho_pT', P, T-273.15);

GLMassFlow = qgl_sc * GasDensity; % Kg/s
OilMassFlow = qo_sc * OilDensity; % Kg/s

MolarFlowGL = GLMassFlow/Mv;         % Kgmol/s
MolarFlowOil = OilMassFlow/Ml;       % Kgmol/s
MolarFlowTot = MolarFlowOil/(1-Fv); % Kgmol/s

end
```

“lee_gonzalez.m”

```
function mu_v = lee_gonzalez(rho, T, Mg)
rho = rho/1000; % From kg/m3 to g/cm3
T = T*1.8; % From Kelvin to Rankine

A1 = (9.379 + 0.01607*Mg)*T^1.5/(209.2 + 19.26*Mg + T);
A2 = 3.448 + 986.4/T + 0.01009*Mg;
A3 = 2.447 - 0.2224*A2;

mu_v = A1 *exp(A2*rho^A3)*10^(-4);
end
```

“getPVTproperties.m”

```
function [ql, qg, LiquidDensity,GasDensity,visc_l,visc_v, SIG, gamma_g,
Cp]=getPVTproperties(T, P, wc, MolarFlow, MolarFlowGL, y_GL)

global dens_o_sc dens_w_sc Fv_sc Ml_sc

NRerror = 1e-2;          % Newton-Rhapson error tolerance
e = 1e-2;                % Fugacity convergence tolerance

% Pl = 37;                 % Lower pressure limit [bar]
% Pu = 100;                % Upper pressure limit [bar]
% dP = 1;                  % Delta pressure [bar]


%for P=Pl:dP:Pu;           % [bar]


convergence_issue = false;

frr=1;                                % Activate the fugacity while-loop

R = 8.314E-5;                           % Gas constant
FluidProp = xlsread ( 'FluidProp' );    % Fluid properties
Pci = FluidProp(:,4);                   % Critical pressure [bar]
Tci = FluidProp(:,5);                   % Critical temperature [K]
Vci = FluidProp(:,6);                   % Critical volume [m3/kmol]
kij = xlsread ( 'PR-EOS-BIPS' );        % BIP's
kij= kij + kij';
z = FluidProp(:,1);                     % Mixture composition
Mi = FluidProp(:,2);                    % Molar weight kg/kmol
AF = FluidProp(:,8);                    % Acentric factor
Tb = FluidProp(:,9);                    % Component boiling point (Rankine)
VTP = FluidProp(:,11);                  % PR Volume translation parameters



MolarFlowTot = MolarFlow + MolarFlowGL; % Now including GL

z = (z*MolarFlow + y_GL*MolarFlowGL)/MolarFlowTot; % New composition due to
GL
z = z/sum(z);  % Make sure the unit is 1.

%% K Value estimate; Wilson equation

Pri = P ./ Pci;                       % Reduced pressure
Tri = T ./ Tci;                       % Reduced temperature
```

```
K = exp(5.37 * (1 + AF) .* (1 - 1 ./ Tri)) ./ Pri ;   % Phase equilibrium
factor

m = zeros(length(z), 1);
for j = 1 : length(z)

    if AF(j) < 0.4

        m(j) = 0.37464 + 1.54226 * AF(j) - 0.26992 * AF(j)^2;

    else

        m(j) = 0.3796 + 1.485 * AF(j) - 0.1644 * AF(j)^2 + 0.01667 *
AF(j)^3;

    end
end

alfa = (1 + m .* (1 - sqrt(Tri))).^2;

a = 0.45724 * R^2 * Tci.^2 ./ Pci .* alfa;

b = 0.07780 * R * Tci ./ Pci;

A = a * P / (R * T)^2;

B = b * P / (R * T);

%% Fv calculation; Rachford-Rice

while frr>e      %Fugacity tolerance constraint


Fvmin = 1 / (1 - max(K));
Fvmax = 1 / (1 - min(K));


%syms Fv        % Matlab numerical solver
%Fv=double(vpasolve(sum(z .* (K - 1) ./ (1 + Fv(1) * (K - 1))),Fv,[Fvmin
Fvmax]));


% Newton-Raphson method
Fv(1) = (Fvmin+Fvmax)/2;            % Initial guess

hFv = sum(z .* (K - 1) ./ (1 + Fv(1) * (K - 1)));

dhFv = - sum(z .* (K - 1).^2 ./ (Fv(1) * (K - 1) + 1).^2);

Fv(2) = Fv(1) - hFv / dhFv;

while abs(Fv(1) - Fv(2)) > NRerror     % Error tolerance

    Fv(1) = Fv (2);
```

```
    hFv = sum(z .* (K - 1) ./ (1 + Fv(1) * (K - 1)));

    dhFv = - sum(z .* (K - 1).^2 ./ (Fv(1) * (K - 1) + 1).^2);

    Fv(2) = Fv(1) - hFv / dhFv;

end

 if Fv(2) > Fvmin && Fv(2) < Fvmax
     Fv = Fv(2);
 else
     Fv = Fv(2);
  %   disp('Fv Unable to converge');
  %   disp(Fv);
     if Fv>1
         Fv = 1;
              convergence_issue = true;
     elseif Fv<0
         Fv = 0;
              convergence_issue = true;
     end
 end

% Liquid and vapor phase composition

x = z ./ (Fv .* (K - 1) + 1);

y = K .* x;


%% A / B parameters

Aij = (1-kij) .* sqrt(A * A');

AL = x' * Aij * x;

AV = y' * Aij * y;

BV = y' * B;

BL = x' * B;

syms Zl Zv

Zl = real(double(solve(Zl^3 - (1 - BL)*Zl^2 + (AL -3*BL^2-2*BL)*Zl-(AL*BL-
BL^2-BL^3))));

Zl = Zl(1);


Zv = real(double(solve(Zv^3 - (1 - BV)*Zv^2 + (AV -3*BV^2-2*BV)*Zv-(AV*BV-
BV^2-BV^3))));

Zv = Zv(1);
```

```
%% Fugacity

% Liquid fugacity
fl = exp(B/BL*(Zl-1)-log(Zl-BL)+AL/(2*sqrt(2)*BL)*(B/BL-
2/AL*Aij*x)*log((Zl+(1+sqrt(2))*BL)/(Zl+(1-sqrt(2))*BL))+log(x*P));

% Vapor fugacity
fv = exp(B/BV*(Zv-1)-log(Zv-BV)+AV/(2*sqrt(2)*BV)*(B/BV-
2/AV*Aij*y)*log((Zv+(1+sqrt(2))*BV)/(Zv+(1-sqrt(2))*BV))+log(y*P));

% Fugacity ratio
fr=fl./fv;

% K value update
K=K.*fr;

% Convergence test
frr=sum((1-fr).^2);

if convergence_issue
    break;
end

end



% Fv<0 => Single phase, liquid

if Fv<0

   % Fv = 0;
    x=z;
    y=zeros(length(z),1);


% Fv>1 => Single phase, vapor

elseif Fv>1;

    %Fv = 1;
    y=z;
    x=zeros(length(z),1);

end

% Save Fv for every pressure calculation in a vector
% Fvv=Fv;

%% Vapor properties


Mv = Mi'*y;                                        % Vapor phase molar weight

%GasDensityEOS = P * Mv*1E-3 / (Zv * R * T);    % EOS calculated vapor
density
```

```
vvEOS = Zv * R * T* 1E+3/P;                          % EOS calculated vapor
molar volume

c = b*1E3 .* VTP;                                    % Volume translation
parameter

vvCOR = vvEOS - sum(c.*y);                          % Corrected vapor molar
volume
GasDensity = Mv / vvCOR;                    % Corrected vapor density



%% Liquid properties

Ml = Mi'*x;                                          % Liquid phase molar
weight

vlEOS = Zl * R * T* 1E+3/P;                          % Liquid phase molar
volume

c = b*1E3 .* VTP;                                    % Volume translation
parameter

vlCOR = vlEOS - sum(c.*x);                          % Corrected liquid phase
molar volume


%OilDensityEOS = P * Ml*1E-3 / (Zl * R * T);    % EOS calculated liquid
density
OilDensity = Ml / vlCOR;                    % Corrected liquid density

WaterDensity = XSteam('rho_pT', P, T-273.15);

GasMassFlow = MolarFlowTot*Mv; % Kg/s
OilMassFlow = MolarFlowTot*Ml; % Kg/s

if sum(y)>0
    qg = GasMassFlow/GasDensity; % m3/s
else
    qg = 0;
end
if sum(x)>0
    qo = OilMassFlow/OilDensity; % m3/s
else
    qo = 0;
end
qo_sc = MolarFlow*(1-Fv_sc)*Ml_sc/dens_o_sc;
qw_sc = wc*qo_sc/(1-wc);
qw = qw_sc*dens_w_sc/XSteam('rho_pT', P, T-273.15);
ql = qo+qw;

LiquidDensity = (WaterDensity*qw + OilDensity*qo)/ql;
gamma_g = Mv/28.97;
gamma_o = OilDensity/dens_w_sc;
gamma_l = (dens_o_sc*qo + dens_w_sc*qw)/(ql*dens_w_sc);

%% Viscosities
a0 = 0.1023;
```

```
a1 = 0.023364;
a2 = 0.058533;
a3 = -0.040758;
a4 = 0.0093324;

% Units should be in: cp, Atm, K.
Pci = Pci*0.986923267;


mu_i = zeros(length(z), 1);
for i =1:length(z)
    xi_Ti = (Tci(i)*Mi(i)^3/Pci(i)^4)^(1/6);
    if Tri(i)<=1.5
        mu_i(i) = 34*10^(-5)*Tri(i)^0.94/xi_Ti;
    else
        mu_i(i) = 17.78*10^(-5)*(4.58*Tri(i)-1.67)^(5/8)/xi_Ti;
    end
end

% Gas:
if sum(y)>0
% Tpc_v = sum(Tci.*y);
% Ppc_v = sum(Pci.*y);
% Vpc_v = sum(Vci.*y);
%
% mu_0_v = sum(mu_i.*y.*sqrt(Mi))/(sum(y.*sqrt(Mi)));
%
% xi_Tv = (Tpc_v/(Mv^3*Ppc_v^4))^(1/6);
% rho_prv = GasDensity/Mv*Vpc_v;
%mu_v = ((a0 + rho_prv*a1 + rho_prv^2*a2 + rho_prv^3*a3 + rho_prv^4*a4)^4-
10^(-4))/xi_Tv + mu_0_v;
visc_v = lee_gonzalez(GasDensity, T, Mv);
else
    visc_v = 0;
%    mu_v2 = 0;
end

% Liquid:
if sum(x)>0
Tpc_l = sum(Tci.*x);
Ppc_l = sum(Pci.*x);
Vpc_l = sum(Vci.*x);

mu_0_l = sum(mu_i.*x.*sqrt(Mi))/(sum(x.*sqrt(Mi)));

xi_Tl = (Tpc_l/(Ml^3*Ppc_l^4))^(1/6);
rho_prl = OilDensity/Ml*Vpc_l;
visc_o = (((a0 + rho_prl*a1 + rho_prl^2*a2 + rho_prl^3*a3 +
rho_prl^4*a4)^4-10^(-4))/xi_Tl + mu_0_l);
else
    visc_o = 0;
end

visc_w = XSteam('my_pT', P, T-273.15)*1000;

if qo/ql>=0.5
    visc_l = visc_o*exp(3.6*(1-qo/ql));
elseif qo/ql<=0.33
    visc_l = (1+2.5*qo/ql*(visc_o+0.4*visc_w)/(visc_o+visc_w))*visc_w;
else
```

```
    visc_weighted1 = visc_o*exp(3.6*(1-qo/ql));
    visc_weighted2 =
(1+2.5*qo/ql*(visc_o+0.4*visc_w)/(visc_o+visc_w))*visc_w;
    visc_l = ((qo/ql-0.33)*visc_weighted1 + (0.5-
qo/ql)*visc_weighted2)/(0.5-0.33);
end

SIG_sum = 0;
for i =1:length(z)
    if Mi(i)>90 % I.e. C7+
        Parachor = 35 + 2.4*Mi(i);
    else
        Parachor = 25.3 + 2.86*Mi(i);
    end
    if sum(y)==0
        SIG_sum = SIG_sum + Parachor*(x(i)*OilDensity/Ml)/1000;
    elseif sum(x)==0
        SIG_sum = SIG_sum + Parachor*(y(i)*GasDensity/Mv)/1000;
    else
        SIG_sum = SIG_sum + Parachor*(x(i)*OilDensity/Ml -
y(i)*GasDensity/Mv)/1000;
    end
end
SIG = SIG_sum^4*0.001;
Tbnoil = Tb'*x;
Tbngas = Tb'*y;

%% Calculation of Cp
Cpw = XSteam('Cp_pT', P, T-273.15)*1000;
 [Cpo, Cpg] = HeatCapacity(gamma_g, T, Tbnoil, Tbngas);
Cp = (OilDensity*qo*Cpo + WaterDensity*qw*Cpw +
GasDensity*qg*Cpg)/(LiquidDensity*ql+GasDensity*qg);
%Cp = 2400;

end

%figure(1)
%plot(Pl:dP:Pu,Fvv*100)
%xlabel('pressure [bar]')
%ylabel('vapor mole fraction [%]')

%figure(2)
%hold on
%plot(Pl:dP:Pu,OilDensityEOS,'r')
%plot(Pl:dP:Pu,OilDensityCOR)
%hold off

%figure(3)
%hold on
%plot(plot(Pl:dP:Pl+dP*(sum(Fvv>0)-1),GasDensityCOR(1:sum(Fvv>0))),'r')
%plot(plot(Pl:dP:Pl+dP*(sum(Fvv>0)-1),GasDensityEOS(1:sum(Fvv>0))))
%hold off
```

“GL_sequential_NR.m” (Compositional model)

```
function [pTubing, pPLEM_WH, pRiser_Flow, p, T, qo_sc_res, qGL_sc_res,
dp_choke] = GL_sequential_NR(MolarFlowTot, y_GL, pR, TR)

global  pSep pBP_0 h_tubing l_tubing h_flowline l_flowline h_riser l_riser
...
        nWells bara day MolarFlowGL_max wc MolarFlowTot_min l_PLEM_WH
h_PLEM_WH ...
        nManifold wells_in_manifold step_length ID_t ID_f ID_m ...
        h_oil h_gas k_steel  ...
         ro_m ro_f h_sea k_ins ro_ins_m ro_ins_f T_sea Fv_sc Ml_sc
dens_o_sc  Mg_sc dens_g_sc


[pWf] = IPR_multi(MolarFlowTot, pBP_0, pR);


% qw_sc = MolarFlowTot.*wc;
% qg_sc = MolarFlowTot.*GOR_0;
MolarFlowTot_init = MolarFlowTot;
fprintf('dp_choke [bar]: \n\n');
step_length = 100; % [m]

%% Pressure drop in tubing, from Bottom of Well to WH.
n_steps_t = zeros(1, nWells);
for j = 1:nWells
    n_steps_t(j) = l_tubing(j)/step_length;    % For tubing
    pTubing = zeros(n_steps_t(j)+1, nWells);
    TTubing = pTubing;
end



MolarFlowGL = zeros(1, nWells);
dp_choke = zeros(1, nWells);
s = 0;
start = true;
%first_time_problem = true;
first_iteration = true;
reduce_init_rate = false;
reduction_trouble = false;
recalculate = false;
endless_loop_count = ones(1, nWells);

while s>=0
    if s ==0
        temp_1 = 1;
    else
        temp_1=s;
    end
while (abs(dp_choke (1, temp_1)) > 0.1) || start
    start = false;
for j = 1:nWells

    if s>0  % "s" indicates the well that is currently being re-calculated
because the pressure is too low.
        j = s;
```

```
        if MolarFlowGL(1, j) == 0 && optimal_oil_rate==false  % I.e. first
iteration.

            dp_choke_old = dp_choke(s);
            MolarFlowGL_old = 0;
            MolarFlowGL(1, j) = MolarFlowGL_max;

        elseif dp_choke(1, j)<0 && MolarFlowGL_old == 0 &&
optimal_oil_rate==false   % I.e. deltaP(q_gl_max) is too low/below zero.

            if MolarFlowGL(1, j) > (MolarFlowGL_max*0.65^2) % I.e. 3
reduced GL rates
                MolarFlowGL(1, j) = MolarFlowGL(1, j)*0.65;
            else
                % Reduce to oil rate until a valid rate is found
(dp_choke>0 with MolarFlowGL_max):

                    MolarFlowTot_old = MolarFlowTot(j);
                    MolarFlowGL(1, j) = MolarFlowGL_max;
                    MolarFlowTot(1, j) = MolarFlowTot(1,
j)*(0.9^(endless_loop_count(s))); % This re-calculates the first deltaP for
no GL and lower oil rate.
                    [pWf(1, j)] = IPR_multi(MolarFlowTot(1, j), pBP_0, pR);
%                     qw_sc(j) = MolarFlowTot(1, j).*wc(j);
%                     qg_sc(j) = MolarFlowTot(1, j).*GOR_0;

                if MolarFlowTot(1, j)< MolarFlowTot_min
                    reduce_init_rate = true;
                    reduce_init_well = j;
                    s = 0;
%                     MolarFlowGL = 0;
%                     MolarFlowTot_init = MolarFlowTot_init*0.99;
%                     MolarFlowTot = MolarFlowTot_init;
                    fprintf('Error: pR is too low for well %d to be
produced. \n Probable reason is that initial rates in all wells are too
high. Rates reduced with 5%%. \n', j);
                    break;
                elseif  pWf(j)<pSep
                    reduce_init_rate = true;
                    reduce_init_well = j;
                    s = 0;
%                     MolarFlowGL = 0;
%                     MolarFlowTot_init = MolarFlowTot_init*0.99;
%                     MolarFlowTot = MolarFlowTot_init;
                    fprintf('Error: pR is too low for well %d to be
produced. \n Probable reason is that initial rates in all wells are too
high. Rates reduced with 5%%. \n', j);
                    break;


                end

                    reduced_oil_rate = true;
                    first_valid_value = true;
            end


        else
```

```
            if reduced_oil_rate  % Solver has found a reduced oil rate that
gives positive dp_choke. MolarFlowGL is now kept the same.
                if first_valid_value  % Solver has found the first valid
reduced oil rate. This is the first iteration. Because it's uncertain what
the valid GL rate is, previous dp_choke's are uncertain. This is calculated
now.
                    MolarFlowGL_old = -1;  % Make the old value of zero
invalid
                    temp_1 = MolarFlowTot(j); % This is the lower boundary
for the oil rate.
                    MolarFlowTot(j) = MolarFlowTot_init(j); %   % Slacker
bounds give more stable converging, but also risk the rate being lowered
too much (lower bound slack) and the runtime to increase (upper bound
slack)          This is the upper boundary for the oil rate =
lower_boundary/0.9
                    MolarFlowTot_old = temp_1;
                    dp_choke_old = dp_choke(j); % Newly calculated,
positive value.
                    count_rate = 0;
                    first_valid_value = false;
                else
                    count_rate = count_rate+1;
                    if count_rate>20
                        fprintf('The solver has an issue with converging
the oil rate for well %d \n\n', s);

                        endless_loop_count(s) = endless_loop_count(s) + 1;

                        reduction_trouble = true;
                        break;
                    end
                    temp_1 = MolarFlowTot(1, j);
                    MolarFlowTot(j) = MolarFlowTot (j) -
dp_choke(j)/(dp_choke(j)-dp_choke_old)*(MolarFlowTot(j)-MolarFlowTot_old);

                    MolarFlowTot_old = temp_1;
                    dp_choke_old = dp_choke(j);
                end
                    [pWf(1, j)] = IPR_multi(MolarFlowTot(1, j), pBP_0, pR);
%                     qw_sc(j) = MolarFlowTot(1, j).*wc(j);
%                     qg_sc(j) = MolarFlowTot(1, j).*GOR_0;



            else

            temp_1 = MolarFlowGL(1, j);

            MolarFlowGL(1, j) = MolarFlowGL(1, j) - dp_choke(1,
j)/(dp_choke(1, j)-dp_choke_old)*(MolarFlowGL(1, j)-MolarFlowGL_old);    %
x = x -f(x)/f'(x)

            dp_choke_old = dp_choke(1, j);
            MolarFlowGL_old = temp_1;

            end

        end
    elseif recalculate
```

```
        s = s2;
        j = s;
    end

    theta = asind(h_tubing(j)/l_tubing(j));   % Inclination in degrees from
horizontal.
    pTubing (1, j) = pWf(j);
    TTubing(1, j) = TR;
    TVD_temp(1, j) = TR;
    ge = (TR-T_sea)/h_tubing(j);  % Geothermal gradient


    for i = 1:n_steps_t(j)




        [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG, gamma_g, Cp] =
getPVTproperties(TTubing(i, j), pTubing(i, j), wc, MolarFlowTot(j),
MolarFlowGL(j), y_GL); % Local fluid rates and properties

        if i == 1
            vm_old = (ql+qg)/(pi * (ID_t/2)^2);
        else
            vm_old = vm;
        end
        vm = (ql+qg)/(pi * (ID_t/2)^2);

        dvds = (vm-vm_old)/step_length;
        [dpds, rho_m, holdup] = multi_dpdx(ql, qg, dens_l, dens_g, visc_l,
visc_g, theta, ID_t, SIG);   % [Pa]
        % Rho_m is with slip
        pTubing(i+1, j) = pTubing(i, j) - dpds/bara*step_length;
        hi = (dens_l*ql*h_oil + dens_g*qg*h_gas)/(dens_l*ql+dens_g*qg);  %
Fluid convection heat transfer coefficient of mixture. Assumes h_oil
=h_water.

        TVD_temp(i+1, j) = TVD_temp(i, j)-ge*step_length*sind(theta);

        Wt = (dens_l*ql+dens_g*qg);

        rho_ns = ql/(ql+qg)*dens_l + qg/(qg+ql)*dens_g;
        TTubing (i+1, j) = multi_dTdx_tubing(TTubing(i, j), pTubing(i+1,
j), theta, rho_m, rho_ns, ge, Cp, holdup, -dpds, dvds, Wt, gamma_g, vm,
TVD_temp(i, j), hi);
%           [a, b] = multi_dTdx_tubing(TTubing(i, j), pTubing(i+1, j),
theta, rho_m, rho_ns, ge, Cp, holdup, -dpds, dvds, Wt, gamma_g, vm,
TVD_temp(i, j), hi);
%
%      TTubing (i+1, j)=a;
%    Utot (i) = b;
        if sum( imag(pTubing(:, j)) )~=0||pTubing(i+1,
j)<pSep||isnan(pTubing(i+1, j))
            reduce_init_rate = true;
            reduce_init_well = j;
            pTubing(:, j) = real(pTubing(:, j));  % If pTubing goes below
zero, imaginary numbers will be generated.
            break;
        end
```

```
         disp(i);
    end
    if reduce_init_rate
        break;
    end
    Cp_Balance_t(j) = Cp*(dens_l*ql+dens_g*qg);
    Temp_Balance_t(j) = Cp_Balance_t(j)*TTubing(n_steps_t(j)+1, j);



   if s>0  % Only calculate lowest well.
       break;
   end

end

if reduce_init_rate == false;
if reduction_trouble
    reduction_trouble = false;
    break;
end
% Find final nonzero element in tubing (pWh)
for i = 1:nWells
    if pTubing(end, i)==0
        X = find (pTubing(:, i) == 0);
        pTubing_end(i) = pTubing( X(1)-1, i);
    else
        pTubing_end(i) = pTubing(end, i);
    end


end




%% Pressure drop in riser and flowline, from Sep to PLEM.
n_steps_f = (l_riser+l_flowline)/step_length;

pRiser_Flow = zeros(n_steps_f+1, 1);
TRiser_Flow = pRiser_Flow;
pRiser_Flow (end) = pSep;

for i = (n_steps_f):-1:1

    [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG] = getPVTproperties(TR,
pRiser_Flow(i+1), wc, sum(MolarFlowTot), sum(MolarFlowGL), y_GL); % Local
fluid rates and properties

    if i<=l_flowline/step_length  % I.e. is the calculation in the flowline
or in the riser
        theta = asind(h_flowline/l_flowline);
    else
        theta = asind(h_riser/l_riser);
    end
```

```
     dpds = multi_dpdx(ql, qg, dens_l, dens_g, visc_l, visc_g, theta, ID_f,
SIG);   % [Pa]
        % Rho_m is with slip (check formulas if this is true)
     pRiser_Flow(i) = pRiser_Flow(i+1) + dpds/bara*step_length;

end


n_steps_m = zeros(1, nManifold);  % Steps from PLEM to Manifold
for j = 1:nManifold
    n_steps_m(j) = l_PLEM_WH(j)/step_length;   % For tubing
end
pPLEM_WH = zeros(  max(n_steps_m)+1, nManifold);
pPLEM_WH(end, :) = pRiser_Flow(1, :);



for r = 1: nManifold   % "r" is a new variabel so not to disturb "j" from
the tubing-section.

    % To keep track of which wells are calculated:
    temp_10=0;
    for y = 1:r
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end

    for i = ( max(n_steps_m) ):-1: ( max(n_steps_m) - n_steps_m(r)+1   )




        [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG] =
getPVTproperties(TR, pPLEM_WH(i+1, r), wc,
sum(MolarFlowTot(temp_9+1:temp_10)), sum(MolarFlowGL(temp_9+1:temp_10)),
y_GL); % Local fluid rates and properties

        theta = asind(h_PLEM_WH(r)/l_PLEM_WH(r));

        dpds = multi_dpdx(ql, qg, dens_l, dens_g, visc_l, visc_g, theta,
ID_m(r), SIG);   % [Pa]

        pPLEM_WH(i, r) = pPLEM_WH(i+1, r) + dpds/bara*step_length;
    end

end

for i = 1:nManifold
    if pPLEM_WH(1, i)==0
        X = find (pPLEM_WH(:, i) == 0);
        Calc_start(i) = X(end)+1;
        pPLEM_WH_end(1, i) = pPLEM_WH( X(end)+1, i);
    else
        pPLEM_WH_end(1, i) = pPLEM_WH(1, i);
        Calc_start(i) = 1;
    end
end
```

```
%% Find temperature in Manifold
TPLEM_WH = zeros(size(pPLEM_WH));
for i = 1:nManifold
    temp_10=0;
    for y = 1:i
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end

    TPLEM_WH(Calc_start(i), i) =
sum(Temp_Balance_t(temp_9+1:temp_10))/sum(Cp_Balance_t(temp_9+1:temp_10));
end

for r = 1: nManifold   % "r" is a new variabel so not to disturb "j" from
the tubing-section.

    % To keep track of which wells are calculated:
    temp_10=0;
    for y = 1:r
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end

    for i = ( max(n_steps_m)  - n_steps_m(r)+1  ): ( max(n_steps_m)  )




        [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG, gamma_g, Cp] =
getPVTproperties(TPLEM_WH(i, r), pPLEM_WH(i, r), wc,
sum(MolarFlowTot(temp_9+1:temp_10)), sum(MolarFlowGL(temp_9+1:temp_10)),
y_GL); % Local fluid rates and properties

        theta = asind(h_PLEM_WH(r)/l_PLEM_WH(r));

         if i == 1
             vm_old = (ql+qg)/(pi * (ID_m(r)/2)^2);
         else
             vm_old = vm;
         end
         vm = (ql+qg)/(pi * (ID_m(r)/2)^2);

         dvds = (vm-vm_old)/step_length;
         [dpds, rho_m, holdup] = multi_dpdx(ql, qg, dens_l, dens_g, visc_l,
visc_g, theta, ID_m(r), SIG);   % [Pa]


         hi = (dens_l*ql*h_oil + dens_g*qg*h_gas)/(dens_l*ql+dens_g*qg);  %
Fluid convection heat transfer coefficient of mixture.
         HTC = 1/( ro_ins_m(r)/(hi*ID_m(r)/2) +
log(2*ro_m(r)/ID_m(r))*ro_ins_m(r)/k_steel ...
             + ro_ins_m(r)*log(ro_ins_m(r)/ro_m(r))/k_ins  + 1/h_sea);
         % Calculation of relaxation distance (Ramsey 1962)
         A = (dens_l*ql+dens_g*qg)*Cp/(2*HTC*pi*ro_ins_m(r));
         rho_ns = ql/(ql+qg)*dens_l + qg/(qg+ql)*dens_g;
         TPLEM_WH(i+1, r) = multi_dTdx_pipe(TPLEM_WH(i, r), pPLEM_WH(i+1,
r), theta, rho_m, rho_ns, 0, Cp, holdup, -dpds, dvds, A, gamma_g, vm,
T_sea);
        %  TPLEM_WH(i+1, r) = TR;
```

```
    end
    Cp_Balance_m(r) = Cp*(dens_l*ql+dens_g*qg);
    Temp_Balance_m(r) = Cp_Balance_m(r)*TPLEM_WH(end, r);

end
TRiser_Flow (1) = sum(Temp_Balance_m(:))/sum(Cp_Balance_m(:));
for i = 1:(n_steps_f)

    [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG, gamma_g, Cp] =
getPVTproperties(TRiser_Flow (i), pRiser_Flow(i), wc, sum(MolarFlowTot),
sum(MolarFlowGL), y_GL); % Local fluid rates and properties

    if i<=l_flowline/step_length  % I.e. is the calculation in the flowline
or in the riser
        theta = asind(h_flowline/l_flowline);
    else
        theta = asind(h_riser/l_riser);
    end

         if i == 1
             vm_old = (ql+qg)/(pi * (ID_f/2)^2);
         else
             vm_old = vm;
         end
         vm = (ql+qg)/(pi * (ID_f/2)^2);

         dvds = (vm-vm_old)/step_length;
         [dpds, rho_m, holdup] = multi_dpdx(ql, qg, dens_l, dens_g, visc_l,
visc_g, theta, ID_f, SIG);   % [Pa]


         hi = (dens_l*ql*h_oil + dens_g*qg*h_gas)/(dens_l*ql+dens_g*qg);  %
Fluid convection heat transfer coefficient of mixture.
         HTC = 1/( ro_ins_f/(hi*ID_f/2) + log(2*ro_f/ID_f)*ro_ins_f/k_steel
...
             + ro_ins_f*log(ro_ins_f/ro_f)/k_ins  + 1/h_sea);
         % Calculation of relaxation distance (Ramsey 1962)
         A = (dens_l*ql+dens_g*qg)*Cp/(2*HTC*pi*ro_ins_f);
         rho_ns = ql/(ql+qg)*dens_l + qg/(qg+ql)*dens_g;
         TRiser_Flow (i+1) = multi_dTdx_pipe(TRiser_Flow (i), pRiser_Flow
(i), theta, rho_m, rho_ns, 0, Cp, holdup, -dpds, dvds, A, gamma_g, vm,
T_sea);
         %TRiser_Flow (i+1) = TR;

end

for i = (n_steps_f):-1:1

    [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG] =
getPVTproperties(TRiser_Flow(i+1), pRiser_Flow(i+1), wc, sum(MolarFlowTot),
sum(MolarFlowGL), y_GL); % Local fluid rates and properties

    if i<=l_flowline/step_length  % I.e. is the calculation in the flowline
or in the riser
        theta = asind(h_flowline/l_flowline);
    else
        theta = asind(h_riser/l_riser);
    end
```

```
     dpds = multi_dpdx(ql, qg, dens_l, dens_g, visc_l, visc_g, theta, ID_f,
SIG);    % [Pa]
        % Rho_m is with slip (check formulas if this is true)
     pRiser_Flow(i) = pRiser_Flow(i+1) + dpds/bara*step_length;

end

pPLEM_WH(end, :) = pRiser_Flow(1, :);



for r = 1: nManifold   % "r" is a new variabel so not to disturb "j" from
the tubing-section.

    % To keep track of which wells are calculated:
    temp_10=0;
    for y = 1:r
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end

    for i = ( max(n_steps_m) ):-1: ( max(n_steps_m) - n_steps_m(r)+1    )




        [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG] =
getPVTproperties(TPLEM_WH(i+1, r), pPLEM_WH(i+1, r), wc,
sum(MolarFlowTot(temp_9+1:temp_10)), sum(MolarFlowGL(temp_9+1:temp_10)),
y_GL); % Local fluid rates and properties

        theta = asind(h_PLEM_WH(r)/l_PLEM_WH(r));

        dpds = multi_dpdx(ql, qg, dens_l, dens_g, visc_l, visc_g, theta,
ID_m(r), SIG);    % [Pa]

        pPLEM_WH(i, r) = pPLEM_WH(i+1, r) + dpds/bara*step_length;
    end

end
```

```
for i = 1:nManifold
    temp_10=0;
    for y = 1:i
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end

    dp_choke(1, temp_9+1:temp_10) = pTubing_end (1, temp_9+1:temp_10) -
pPLEM_WH_end(1, i);
end

disp(dp_choke);
end

if s>0
    if recalculate
        recalculate = false;
        MolarFlowGL_old = -1;
        if dp_choke(s)>0 % Recalculated well with positive choke, initial
rates and no GL
            break;
        end
    end
    temp_1 = s;
else
    break;
end



end


if s>0
    if endless_loop_count(s)==10
        fprintf('Program run into an endless loop of re-iteration for well
%d. Program stops and returns values close to target. \n\n', s);
       break;
    end
end


disp(dp_choke);
    if isnan(sum(dp_choke))||reduce_init_rate % Imaginary numbers due to
high oil rate
        reduce_init_rate = false;
        s = 0;
        %MolarFlowGL(1, 1:nWells) = MolarFlowGL_max;
        start = true;
        if MolarFlowTot_init(reduce_init_well)== 0
            MolarFlowGL(reduce_init_well) = MolarFlowGL_max;
        else
            MolarFlowTot_init(reduce_init_well) =
MolarFlowTot_init(reduce_init_well)*0.9;
        end

        MolarFlowTot = MolarFlowTot_init;
        [pWf] = IPR_multi(MolarFlowTot, pBP_0, pR);
```

```
%         qw_sc = MolarFlowTot.*wc;
%         qg_sc = MolarFlowTot.*GOR_0;
        %fprintf('Error: Too high rates in some of the wells result in NaN.
All rates reduced by 1%%. \n');
    elseif first_iteration
        first_iteration = false;
        if dp_choke<0 % For all wells
            MolarFlowGL (1, 1:nWells) = MolarFlowGL_max;
        end
    else



if (dp_choke+0.1)>=0   % 0.1 is the error margin. If dpChoke>0 then no GL.
    count = [];
    for i = 1:nWells
        if dp_choke(i)>0.1
            if (MolarFlowGL(i)>0 || MolarFlowTot(i)<MolarFlowTot_init(i)-
10^(-10))
                count = [count i];
            end
        end
    end

    if isempty(count)
        break;
    else
        reduced_oil_rate = false;
        optimal_oil_rate = false;

        s = find(  dp_choke(:) == max(dp_choke  (count)));   % S equals the
well with the lowest pressure at WH.
        if length(s)>1
            s = s(1);    % If multiple wells have equal lowst pressure at
WH, start with first one.
        end
        MolarFlowGL_old = -1;  % Wrong value to get the solver started.
        MolarFlowGL(s) = 0; % Calculations starting with reduced rates.
        MolarFlowTot(s) = MolarFlowTot_init(s);
        [pWf(1, s)] = IPR_multi(MolarFlowTot(1, s), pBP_0, pR);
%         qw_sc(s) = MolarFlowTot(1, s).*wc(s);
%         qg_sc(s) = MolarFlowTot(1, s).*GOR_0;
        s2 = s;
        s = 0;
        recalculate = true;
        start = true;
    end
else
    reduced_oil_rate = false;
    optimal_oil_rate = false;

    s = find(  dp_choke(:) == min(dp_choke  (:)));   % S equals the well
with the lowest pressure at WH.
    if length(s)>1
        s = s(1);    % If multiple wells have equal lowst pressure at WH,
start with first one.
    end
        MolarFlowGL_old = -1;  % Wrong value to get the solver started.
        if MolarFlowGL(s)>0 % I.e. re-calculation of GL-rate
            MolarFlowGL(s) = 0; % Calculations starting with reduced rates.
```

```
            MolarFlowTot(s) = MolarFlowTot_init(s);
            [pWf(1, s)] = IPR_multi(MolarFlowTot(1, s), pBP_0, pR);
%             qw_sc(s) = MolarFlowTot(1, s).*wc(s);
%             qg_sc(s) = MolarFlowTot(1, s).*GOR_0;
            s2 = s;
            s = 0;
            recalculate = true;
            start = true;
        end
end

if  s>0 % The previous calculation resultet in infeasible solution (close
to correct solution), but will continue in an endless loop. This answer is
anyway close to the solution.
    if endless_loop_count(s)==1
        endless_loop_count(:) = 1;
    end

end
    end

end




%% Tidy-up section

% Pressures and Temperatures in the pipes:
p = zeros( (length(pTubing(:, 1))+length(pPLEM_WH(:,
1))+length(pRiser_Flow(:, 1)))-1, nWells);
p(1:length(pTubing), :) = pTubing;

T = zeros( (length(pTubing(:, 1))+length(pPLEM_WH(:,
1))+length(pRiser_Flow(:, 1)))-1, nWells);
T(1:length(pTubing), :) = TTubing;

for i = 1:nManifold
    temp_10=0;
    for y = 1:i
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end
    for a = temp_9+1:temp_10
        p( (length(pTubing(:, 1))+1):(length(pTubing(:,
1))+length(pPLEM_WH(:, 1))) , a) = pPLEM_WH(:, i);
        T( (length(pTubing(:, 1))+1):(length(pTubing(:,
1))+length(pPLEM_WH(:, 1))) , a) = TPLEM_WH(:, i);
    end

end

for j =1:nWells
p ( (length(pTubing(:, 1))+length(pPLEM_WH(:, 1))):(length(pTubing(:,
1))+length(pPLEM_WH(:, 1))+length(pRiser_Flow(:, 1)))-1, j) = pRiser_Flow;
T ( (length(pTubing(:, 1))+length(pPLEM_WH(:, 1))):(length(pTubing(:,
1))+length(pPLEM_WH(:, 1))+length(pRiser_Flow(:, 1)))-1, j) = TRiser_Flow;
end
```

```
% GL rates:
for j =1:nWells
    temp = find(MolarFlowGL(:, j) ~= 0);
    if isempty(temp)
        MolarFlowGL_res(1, j) = 0;
    else
        MolarFlowGL_res(1, j) = MolarFlowGL(temp(end), j);
    end
end
MolarFlowGL_res = MolarFlowGL_res*day;

% Oil rates:
size_oil = size(MolarFlowTot);
if size_oil(1)>1 % I.e. at least one well is calculated with reduced oil
rate.
    MolarFlowTot_res = zeros(1, nWells);
    for k = 1:nWells
        nonzero_value = find(MolarFlowTot(:, k) ~=0);
        correct_value = nonzero_value(end);
        MolarFlowTot_res(k) = MolarFlowTot(correct_value, k)*day;
    end
else
    MolarFlowTot_res = MolarFlowTot*day;   % I.e. Original oil rates count.
end

qo_sc_res = MolarFlowTot_res*(1-Fv_sc)*Ml_sc/dens_o_sc;
qGL_sc_res = MolarFlowGL_res*Mg_sc/dens_g_sc;


fprintf('Calculated oil rates [Sm3/D]: \n\n');
format short g;
disp(qo_sc_res);
fprintf('Calculated GL rates [Sm3/D]: \n\n');
disp(qGL_sc_res);


end
```

“GL_sequential_NR.m” (Mecanistic Model)

```
function [pTubing, pPLEM_WH, pRiser_Flow, p, T, qo_sc_res, qgl_sc_res,
dp_choke] = GL_sequential_NR(qo_sc, pR, TR)

global  pSep pBP_0 h_tubing l_tubing h_flowline l_flowline h_riser l_riser
...
        nWells bara day qgl_sc_max wc GOR_0 qo_sc_min l_PLEM_WH h_PLEM_WH
...
        nManifold wells_in_manifold step_length ID_t ID_f ID_m ...
        h_oil h_gas k_steel  ...
         ro_m ro_f h_sea k_ins ro_ins_m ro_ins_f T_sea


[pWf] = IPR_multi(qo_sc, pBP_0, pR);


qw_sc = qo_sc.*wc;
qg_sc = qo_sc.*GOR_0;
qo_sc_init = qo_sc;
fprintf('dp_choke [bar]: \n\n');
step_length = 10; % [m]

%% Pressure drop in tubing, from Bottom of Well to WH.
n_steps_t = zeros(1, nWells);
for j = 1:nWells
    n_steps_t(j) = l_tubing(j)/step_length;   % For tubing
    pTubing = zeros(n_steps_t(j)+1, nWells);
    TTubing = pTubing;
end



qgl_sc = zeros(1, nWells);
dp_choke = zeros(1, nWells);
s = 0;
start = true;
%first_time_problem = true;
first_iteration = true;
reduce_init_rate = false;
reduction_trouble = false;
recalculate = false;
endless_loop_count = ones(1, nWells);

while s>=0
    if s ==0
        temp_1 = 1;
    else
        temp_1=s;
    end
while (abs(dp_choke (1, temp_1)) > 0.1) || start
    start = false;
for j = 1:nWells

    if s>0  % "s" indicates the well that is currently being re-calculated
because the pressure is too low.
        j = s;
```

```
        if qgl_sc(1, j) == 0 && optimal_oil_rate==false  % I.e. first
iteration.

            dp_choke_old = dp_choke(s);
            qgl_sc_old = 0;
            qgl_sc(1, j) = qgl_sc_max;

        elseif dp_choke(1, j)<0 && qgl_sc_old == 0 &&
optimal_oil_rate==false   % I.e. deltaP(q_gl_max) is too low/below zero.

            if qgl_sc(1, j) > (qgl_sc_max*0.65^2) % I.e. 3 reduced GL rates
                qgl_sc(1, j) = qgl_sc(1, j)*0.65;
            else
                % Reduce to oil rate until a valid rate is found
(dp_choke>0 with qgl_sc_max):

                    qo_sc_old = qo_sc(j);
                    qgl_sc(1, j) = qgl_sc_max;
                    qo_sc(1, j) = qo_sc(1,
j)*(0.9^(endless_loop_count(s))); % This re-calculates the first deltaP for
no GL and lower oil rate.
                    [pWf(1, j)] = IPR_multi(qo_sc(1, j), pBP_0, pR);
                    qw_sc(j) = qo_sc(1, j).*wc(j);
                    qg_sc(j) = qo_sc(1, j).*GOR_0;

                if qo_sc(1, j)< qo_sc_min || pWf(j)<pSep
                    reduce_init_rate = true;
                    reduce_init_well = j;
                    s = 0;
%                     qgl_sc = 0;
%                     qo_sc_init = qo_sc_init*0.99;
%                     qo_sc = qo_sc_init;
                    fprintf('Error: pR is too low for well %d to be
produced. \n Probable reason is that initial rates in all wells are too
high. Rates reduced with 5%%. \n', j);
                    break;
                end

                    reduced_oil_rate = true;
                    first_valid_value = true;
            end


        else

            if reduced_oil_rate  % Solver has found a reduced oil rate that
gives positive dp_choke. qgl_sc is now kept the same.
                if first_valid_value  % Solver has found the first valid
reduced oil rate. This is the first iteration. Because it's uncertain what
the valid GL rate is, previous dp_choke's are uncertain. This is calculated
now.
                    qgl_sc_old = -1;  % Make the old value of zero invalid
                    temp_1 = qo_sc(j); % This is the lower boundary for the
oil rate.
                    qo_sc(j) = qo_sc_init(j); %   % Slacker bounds give
more stable converging, but also risk the rate being lowered too much
(lower bound slack) and the runtime to increase (upper bound slack)
This is the upper boundary for the oil rate = lower_boundary/0.9
                    qo_sc_old = temp_1;
```

```
                    dp_choke_old = dp_choke(j); % Newly calculated,
positive value.
                    count_rate = 0;
                    first_valid_value = false;
                else
                    count_rate = count_rate+1;
                    if count_rate>20
                        fprintf('The solver has an issue with converging
the oil rate for well %d \n\n', s);

                        endless_loop_count(s) = endless_loop_count(s) + 1;

                        reduction_trouble = true;
                        break;
                    end
                    temp_1 = qo_sc(1, j);
                    qo_sc(j) = qo_sc (j) - dp_choke(j)/(dp_choke(j)-
dp_choke_old)*(qo_sc(j)-qo_sc_old);

                    qo_sc_old = temp_1;
                    dp_choke_old = dp_choke(j);
                end
                    [pWf(1, j)] = IPR_multi(qo_sc(1, j), pBP_0, pR);
                    qw_sc(j) = qo_sc(1, j).*wc(j);
                    qg_sc(j) = qo_sc(1, j).*GOR_0;



            else

            temp_1 = qgl_sc(1, j);

            qgl_sc(1, j) = qgl_sc(1, j) - dp_choke(1, j)/(dp_choke(1, j)-
dp_choke_old)*(qgl_sc(1, j)-qgl_sc_old);    % x = x -f(x)/f'(x)

            dp_choke_old = dp_choke(1, j);
            qgl_sc_old = temp_1;

            end

        end
    elseif recalculate
        s = s2;
        j = s;
    end

    theta = asind(h_tubing(j)/l_tubing(j));   % Inclination in degrees from
horizontal.
    pTubing (1, j) = pWf(j);
    TTubing(1, j) = TR;
    TVD_temp(1, j) = TR;
    ge = (TR-T_sea)/h_tubing(j);  % Geothermal gradient


    for i = 1:n_steps_t(j)
```

```
        [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG, gamma_g, Cp] =
local_properties(pTubing(i, j), TTubing(i, j), qo_sc(j), qw_sc(j),
qg_sc(j), qgl_sc(j)); % Local fluid rates and properties

        if i == 1
            vm_old = (ql+qg)/(pi * (ID_t/2)^2);
        else
            vm_old = vm;
        end
        vm = (ql+qg)/(pi * (ID_t/2)^2);

        dvds = (vm-vm_old)/step_length;
        [dpds, rho_m, holdup] = multi_dpdx(ql, qg, dens_l, dens_g, visc_l,
visc_g, theta, ID_t, SIG);   % [Pa]
       % Rho_m is with slip
        pTubing(i+1, j) = pTubing(i, j) - dpds/bara*step_length;
        hi = (dens_l*ql*h_oil + dens_g*qg*h_gas)/(dens_l*ql+dens_g*qg);  %
Fluid convection heat transfer coefficient of mixture. Assumes h_oil
=h_water.

        TVD_temp(i+1, j) = TVD_temp(i, j)-ge*step_length*sind(theta);

        Wt = (dens_l*ql+dens_g*qg);

        rho_ns = ql/(ql+qg)*dens_l + qg/(qg+ql)*dens_g;
        TTubing (i+1, j) = multi_dTdx_tubing(TTubing(i, j), pTubing(i+1,
j), theta, rho_m, rho_ns, ge, Cp, holdup, -dpds, dvds, Wt, gamma_g, vm,
TVD_temp(i, j), hi);
%           [a, b] = multi_dTdx_tubing(TTubing(i, j), pTubing(i+1, j),
theta, rho_m, rho_ns, ge, Cp, holdup, -dpds, dvds, Wt, gamma_g, vm,
TVD_temp(i, j), hi);
%
%      TTubing (i+1, j)=a;
%    Utot (i) = b;
        if sum( imag(pTubing(:, j)) )~=0||pTubing(i+1,
j)<pSep||isnan(pTubing(i+1, j))
            reduce_init_rate = true;
            reduce_init_well = j;
            pTubing(:, j) = real(pTubing(:, j));  % If pTubing goes below
zero, imaginary numbers will be generated.
            break;
        end

    end
    if reduce_init_rate
        break;
    end
    Cp_Balance_t(j) = Cp*(dens_l*ql+dens_g*qg);
    Temp_Balance_t(j) = Cp_Balance_t(j)*TTubing(n_steps_t(j)+1, j);



   if s>0  % Only calculate lowest well.
       break;
   end

end
```

```
if reduce_init_rate == false;
if reduction_trouble
    reduction_trouble = false;
    break;
end
% Find final nonzero element in tubing (pWh)
for i = 1:nWells
    if pTubing(end, i)==0
        X = find (pTubing(:, i) == 0);
        pTubing_end(i) = pTubing( X(1)-1, i);
    else
        pTubing_end(i) = pTubing(end, i);
    end


end
```

```
%% Pressure drop in riser and flowline, from Sep to PLEM.
n_steps_f = (l_riser+l_flowline)/step_length;

pRiser_Flow = zeros(n_steps_f+1, 1);
TRiser_Flow = pRiser_Flow;
pRiser_Flow (end) = pSep;

for i = (n_steps_f):-1:1

    [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG] =
local_properties(pRiser_Flow(i+1), TR, sum(qo_sc), sum(qw_sc), sum(qg_sc),
sum(qgl_sc)); % Local fluid rates and properties

    if i<=l_flowline/step_length  % I.e. is the calculation in the flowline
or in the riser
        theta = asind(h_flowline/l_flowline);
    else
        theta = asind(h_riser/l_riser);
    end

     dpds = multi_dpdx(ql, qg, dens_l, dens_g, visc_l, visc_g, theta, ID_f,
SIG);   % [Pa]
        % Rho_m is with slip (check formulas if this is true)
     pRiser_Flow(i) = pRiser_Flow(i+1) + dpds/bara*step_length;

end


n_steps_m = zeros(1, nManifold);  % Steps from PLEM to Manifold
for j = 1:nManifold
    n_steps_m(j) = l_PLEM_WH(j)/step_length;   % For tubing
end
pPLEM_WH = zeros(  max(n_steps_m)+1, nManifold);
pPLEM_WH(end, :) = pRiser_Flow(1, :);
```

```
for r = 1: nManifold   % "r" is a new variabel so not to disturb "j" from
the tubing-section.

    % To keep track of which wells are calculated:
    temp_10=0;
    for y = 1:r
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end

    for i = ( max(n_steps_m) ):-1: ( max(n_steps_m) - n_steps_m(r)+1   )




        [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG] =
local_properties(pPLEM_WH(i+1, r), TR, sum(qo_sc(temp_9+1:temp_10)),
sum(qw_sc(temp_9+1:temp_10)), sum(qg_sc(temp_9+1:temp_10)),
sum(qgl_sc(temp_9+1:temp_10))); % Local fluid rates and properties

        theta = asind(h_PLEM_WH(r)/l_PLEM_WH(r));

        dpds = multi_dpdx(ql, qg, dens_l, dens_g, visc_l, visc_g, theta,
ID_m(r), SIG);   % [Pa]

        pPLEM_WH(i, r) = pPLEM_WH(i+1, r) + dpds/bara*step_length;
    end

end

for i = 1:nManifold
    if pPLEM_WH(1, i)==0
        X = find (pPLEM_WH(:, i) == 0);
        Calc_start(i) = X(end)+1;
        pPLEM_WH_end(1, i) = pPLEM_WH( X(end)+1, i);
    else
        pPLEM_WH_end(1, i) = pPLEM_WH(1, i);
        Calc_start(i) = 1;
    end
end

%% Find temperature in Manifold
TPLEM_WH = zeros(size(pPLEM_WH));
for i = 1:nManifold
    temp_10=0;
    for y = 1:i
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end

    TPLEM_WH(Calc_start(i), i) =
sum(Temp_Balance_t(temp_9+1:temp_10))/sum(Cp_Balance_t(temp_9+1:temp_10));
end

for r = 1: nManifold   % "r" is a new variabel so not to disturb "j" from
the tubing-section.

    % To keep track of which wells are calculated:
```

```
    temp_10=0;
    for y = 1:r
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end

    for i = ( max(n_steps_m)  - n_steps_m(r)+1  ): ( max(n_steps_m)  )




        [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG, gamma_g, Cp] =
local_properties(pPLEM_WH(i, r), TPLEM_WH(i, r),
sum(qo_sc(temp_9+1:temp_10)), sum(qw_sc(temp_9+1:temp_10)),
sum(qg_sc(temp_9+1:temp_10)), sum(qgl_sc(temp_9+1:temp_10))); % Local fluid
rates and properties

        theta = asind(h_PLEM_WH(r)/l_PLEM_WH(r));

         if i == 1
             vm_old = (ql+qg)/(pi * (ID_m(r)/2)^2);
         else
             vm_old = vm;
         end
         vm = (ql+qg)/(pi * (ID_m(r)/2)^2);

         dvds = (vm-vm_old)/step_length;
         [dpds, rho_m, holdup] = multi_dpdx(ql, qg, dens_l, dens_g, visc_l,
visc_g, theta, ID_m(r), SIG);   % [Pa]


         hi = (dens_l*ql*h_oil + dens_g*qg*h_gas)/(dens_l*ql+dens_g*qg);  %
Fluid convection heat transfer coefficient of mixture.
         HTC = 1/( ro_ins_m(r)/(hi*ID_m(r)/2) +
log(2*ro_m(r)/ID_m(r))*ro_ins_m(r)/k_steel ...
             + ro_ins_m(r)*log(ro_ins_m(r)/ro_m(r))/k_ins  + 1/h_sea);
         % Calculation of relaxation distance (Ramsey 1962)
         A = (dens_l*ql+dens_g*qg)*Cp/(2*HTC*pi*ro_ins_m(r));
         rho_ns = ql/(ql+qg)*dens_l + qg/(qg+ql)*dens_g;
         TPLEM_WH(i+1, r) = multi_dTdx_pipe(TPLEM_WH(i, r), pPLEM_WH(i+1,
r), theta, rho_m, rho_ns, 0, Cp, holdup, -dpds, dvds, A, gamma_g, vm,
T_sea);
        %  TPLEM_WH(i+1, r) = TR;
    end
    Cp_Balance_m(r) = Cp*(dens_l*ql+dens_g*qg);
    Temp_Balance_m(r) = Cp_Balance_m(r)*TPLEM_WH(end, r);

end
TRiser_Flow (1) = sum(Temp_Balance_m(:))/sum(Cp_Balance_m(:));
for i = 1:(n_steps_f)

    [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG, gamma_g, Cp] =
local_properties(pRiser_Flow(i), TRiser_Flow (i), sum(qo_sc), sum(qw_sc),
sum(qg_sc), sum(qgl_sc)); % Local fluid rates and properties

    if i<=l_flowline/step_length  % I.e. is the calculation in the flowline
or in the riser
        theta = asind(h_flowline/l_flowline);
    else
```

```
        theta = asind(h_riser/l_riser);
    end

         if i == 1
             vm_old = (ql+qg)/(pi * (ID_f/2)^2);
         else
             vm_old = vm;
         end
         vm = (ql+qg)/(pi * (ID_f/2)^2);

         dvds = (vm-vm_old)/step_length;
         [dpds, rho_m, holdup] = multi_dpdx(ql, qg, dens_l, dens_g, visc_l,
visc_g, theta, ID_f, SIG);   % [Pa]


         hi = (dens_l*ql*h_oil + dens_g*qg*h_gas)/(dens_l*ql+dens_g*qg);  %
Fluid convection heat transfer coefficient of mixture.
         HTC = 1/( ro_ins_f/(hi*ID_f/2) + log(2*ro_f/ID_f)*ro_ins_f/k_steel
...
             + ro_ins_f*log(ro_ins_f/ro_f)/k_ins  + 1/h_sea);
         % Calculation of relaxation distance (Ramsey 1962)
         A = (dens_l*ql+dens_g*qg)*Cp/(2*HTC*pi*ro_ins_f);
         rho_ns = ql/(ql+qg)*dens_l + qg/(qg+ql)*dens_g;
         TRiser_Flow (i+1) = multi_dTdx_pipe(TRiser_Flow (i), pRiser_Flow
(i), theta, rho_m, rho_ns, 0, Cp, holdup, -dpds, dvds, A, gamma_g, vm,
T_sea);
         %TRiser_Flow (i+1) = TR;

end

for i = (n_steps_f):-1:1

    [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG] =
local_properties(pRiser_Flow(i+1), TRiser_Flow(i+1), sum(qo_sc),
sum(qw_sc), sum(qg_sc), sum(qgl_sc)); % Local fluid rates and properties

    if i<=l_flowline/step_length  % I.e. is the calculation in the flowline
or in the riser
        theta = asind(h_flowline/l_flowline);
    else
        theta = asind(h_riser/l_riser);
    end

     dpds = multi_dpdx(ql, qg, dens_l, dens_g, visc_l, visc_g, theta, ID_f,
SIG);   % [Pa]
        % Rho_m is with slip (check formulas if this is true)
     pRiser_Flow(i) = pRiser_Flow(i+1) + dpds/bara*step_length;

end

pPLEM_WH(end, :) = pRiser_Flow(1, :);



for r = 1: nManifold   % "r" is a new variabel so not to disturb "j" from
the tubing-section.

    % To keep track of which wells are calculated:
```

```
    temp_10=0;
    for y = 1:r
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end

    for i = ( max(n_steps_m) ):-1: ( max(n_steps_m) - n_steps_m(r)+1    )




        [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG] =
local_properties(pPLEM_WH(i+1, r), TPLEM_WH(i+1, r),
sum(qo_sc(temp_9+1:temp_10)), sum(qw_sc(temp_9+1:temp_10)),
sum(qg_sc(temp_9+1:temp_10)), sum(qgl_sc(temp_9+1:temp_10))); % Local fluid
rates and properties

        theta = asind(h_PLEM_WH(r)/l_PLEM_WH(r));

        dpds = multi_dpdx(ql, qg, dens_l, dens_g, visc_l, visc_g, theta,
ID_m(r), SIG);   % [Pa]

        pPLEM_WH(i, r) = pPLEM_WH(i+1, r) + dpds/bara*step_length;
    end

end
```

```
for i = 1:nManifold
    temp_10=0;
    for y = 1:i
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end

    dp_choke(1, temp_9+1:temp_10) = pTubing_end (1, temp_9+1:temp_10) -
pPLEM_WH_end(1, i);
end



end
```

```
if s>0
    if recalculate
        recalculate = false;
        qgl_sc_old = -1;
        if dp_choke(s)>0 % Recalculated well with positive choke, initial
rates and no GL
            break;
        end
    end
    temp_1 = s;
else
    break;
end



end


if s>0
    if endless_loop_count(s)==10
        fprintf('Program run into an endless loop of re-iteration for well
%d. Program stops and returns values close to target. \n\n', s);
       break;
    end
end


disp(dp_choke);
    if isnan(sum(dp_choke))||reduce_init_rate % Imaginary numbers due to
high oil rate
        reduce_init_rate = false;
        s = 0;
        %qgl_sc(1, 1:nWells) = qgl_sc_max;
        start = true;
        if qo_sc_init(reduce_init_well)== 0
            qgl_sc(reduce_init_well) = qgl_sc_max;
        else
            qo_sc_init(reduce_init_well) =
qo_sc_init(reduce_init_well)*0.9;
        end

        qo_sc = qo_sc_init;
        [pWf] = IPR_multi(qo_sc, pBP_0, pR);
        qw_sc = qo_sc.*wc;
        qg_sc = qo_sc.*GOR_0;
        %fprintf('Error: Too high rates in some of the wells result in NaN.
All rates reduced by 1%%. \n');
    elseif first_iteration
        first_iteration = false;
        if dp_choke<0 % For all wells
            qgl_sc (1, 1:nWells) = qgl_sc_max;
        end
    else



if (dp_choke+0.1)>=0   % 0.1 is the error margin. If dpChoke>0 then no GL.
    count = [];
```

```
    for i = 1:nWells
        if dp_choke(i)>0.1
            if (qgl_sc(i)>0 || qo_sc(i)<qo_sc_init(i)-10^(-10))
                count = [count i];
            end
        end
    end

    if isempty(count)
        break;
    else
        reduced_oil_rate = false;
        optimal_oil_rate = false;

        s = find(  dp_choke(:) == max(dp_choke  (count)));   % S equals the
well with the lowest pressure at WH.
        if length(s)>1
            s = s(1);    % If multiple wells have equal lowst pressure at
WH, start with first one.
        end
        qgl_sc_old = -1;  % Wrong value to get the solver started.
        qgl_sc(s) = 0; % Calculations starting with reduced rates.
        qo_sc(s) = qo_sc_init(s);
        [pWf(1, s)] = IPR_multi(qo_sc(1, s), pBP_0, pR);
        qw_sc(s) = qo_sc(1, s).*wc(s);
        qg_sc(s) = qo_sc(1, s).*GOR_0;
        s2 = s;
        s = 0;
        recalculate = true;
        start = true;
    end
else
    reduced_oil_rate = false;
    optimal_oil_rate = false;

    s = find(  dp_choke(:) == min(dp_choke  (:)));   % S equals the well
with the lowest pressure at WH.
    if length(s)>1
        s = s(1);    % If multiple wells have equal lowst pressure at WH,
start with first one.
    end
        qgl_sc_old = -1;  % Wrong value to get the solver started.
        if qgl_sc(s)>0 % I.e. re-calculation of GL-rate
            qgl_sc(s) = 0; % Calculations starting with reduced rates.
            qo_sc(s) = qo_sc_init(s);
            [pWf(1, s)] = IPR_multi(qo_sc(1, s), pBP_0, pR);
            qw_sc(s) = qo_sc(1, s).*wc(s);
            qg_sc(s) = qo_sc(1, s).*GOR_0;
            s2 = s;
            s = 0;
            recalculate = true;
            start = true;
        end
end

if  s>0 % The previous calculation resultet in infeasible solution (close
to correct solution), but will continue in an endless loop. This answer is
anyway close to the solution.
    if endless_loop_count(s)==1
        endless_loop_count(:) = 1;
    end
```

```
end
    end

end




%% Tidy-up section

% Pressures and Temperatures in the pipes:
p = zeros( (length(pTubing(:, 1))+length(pPLEM_WH(:,
1))+length(pRiser_Flow(:, 1)))-1, nWells);
p(1:length(pTubing), :) = pTubing;

T = zeros( (length(pTubing(:, 1))+length(pPLEM_WH(:,
1))+length(pRiser_Flow(:, 1)))-1, nWells);
T(1:length(pTubing), :) = TTubing;

for i = 1:nManifold
    temp_10=0;
    for y = 1:i
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end
    for a = temp_9+1:temp_10
        p( (length(pTubing(:, 1))+1):(length(pTubing(:,
1))+length(pPLEM_WH(:, 1))) , a) = pPLEM_WH(:, i);
        T( (length(pTubing(:, 1))+1):(length(pTubing(:,
1))+length(pPLEM_WH(:, 1))) , a) = TPLEM_WH(:, i);
    end

end

for j =1:nWells
p ( (length(pTubing(:, 1))+length(pPLEM_WH(:, 1))):(length(pTubing(:,
1))+length(pPLEM_WH(:, 1))+length(pRiser_Flow(:, 1)))-1, j) = pRiser_Flow;
T ( (length(pTubing(:, 1))+length(pPLEM_WH(:, 1))):(length(pTubing(:,
1))+length(pPLEM_WH(:, 1))+length(pRiser_Flow(:, 1)))-1, j) = TRiser_Flow;
end

% GL rates:
for j =1:nWells
    temp = find(qgl_sc(:, j) ~= 0);
    if isempty(temp)
        qgl_sc_res(1, j) = 0;
    else
        qgl_sc_res(1, j) = qgl_sc(temp(end), j);
    end
end
qgl_sc_res = qgl_sc_res*day;

% Oil rates:
size_oil = size(qo_sc);
if size_oil(1)>1 % I.e. at least one well is calculated with reduced oil
rate.
    qo_sc_res = zeros(1, nWells);
```

```
    for k = 1:nWells
        nonzero_value = find(qo_sc(:, k) ~=0);
        correct_value = nonzero_value(end);
        qo_sc_res(k) = qo_sc(correct_value, k)*day;
    end
else
    qo_sc_res = qo_sc*day;   % I.e. Original oil rates count.
end

fprintf('Calculated oil rates [Sm3/D]: \n\n');
format short g;
disp(qo_sc_res);
fprintf('Calculated GL rates [Sm3/D]: \n\n');
disp(qgl_sc_res);


end
```

“GL_parallel_NR.m” (Beggs and Brill)

```
function [pTubing, pPLEM_WH, pRiser_Flow, p, qo_sc_res, qg_sc, qw_sc,
qgl_sc_res, dp_choke] = GL_parallel_NR(qo_sc, qw_sc, qg_sc, pR)

global TR pSep pBP_0 h_tubing l_tubing h_flowline l_flowline h_riser
l_riser ...
        nWells bara day wc GOR_0 l_PLEM_WH h_PLEM_WH ...
        nManifold wells_in_manifold step_length qgl_sc_max ID_t ID_f ID_m
  qo_sc_init = qo_sc;
[pWf] = IPR_multi(qo_sc, pBP_0, pR);
qw_sc = qo_sc.*wc;
qg_sc = qo_sc.*GOR_0;
fprintf('dp_choke [bar]: \n\n');
step_length = 10; % [m]

%% Pressure drop in tubing, from Bottom of Well to WH.
n_steps_t = zeros(1, nWells);
for j = 1:nWells
    n_steps_t(j) = l_tubing(j)/step_length;   % For tubing
    pTubing = zeros(n_steps_t(j)+1, nWells);
end



qgl_sc = zeros(1, nWells);
dp_choke = zeros(1, nWells);
epsilon = 100;
increased_init_gl = [];
first_iteration = true;
real_dp_choke = true;
dp_choke_mat = zeros (nWells);
active_wells = 1: nWells;
re_original_rate = false;
    endless_loop = false;
for i = 1:nWells
    reduced_rate (i) = false;
    inactive_wells (i) = false;
    factor(i) = 1;
    original_rate (i) = false;

end

while epsilon > 0.1

for z = [active_wells, nWells+1]


for j = 1:nWells


    theta = asind(h_tubing(j)/l_tubing(j));   % Inclination in degrees from
horizontal.
    pTubing (1, j) = pWf(j);
    T = TR;  % Should be made into a function.

        if pTubing(1, j)<pSep
```

```
  %          pTubing(:, j) = real(pTubing(:, j));  % If pTubing goes below
zero, imaginary numbers will be generated.
            endless_loop = true;
            endless_loop_well = j;
            break;
        end


    for i = 1:n_steps_t(j)




         [ql, qg, dens_l, dens_g, visc_l, visc_g] =
local_properties(pTubing(i, j), T, qo_sc(j), qw_sc(j), qg_sc(j),
qgl_sc(j)); % Local fluid rates and properties

         dpds = dpBeggs_Brill(ql, qg, dens_l, dens_g, visc_l, visc_g,
theta, step_length, ID_t);   % [Pa]

         pTubing(i+1, j) = pTubing(i, j) - dpds/bara;

        if sum( imag(pTubing(:, j)) )~=0 || pTubing(i+1,
j)<pSep||isnan(pTubing(i+1, j))
            pTubing(:, j) = real(pTubing(:, j));  % If pTubing goes below
zero, imaginary numbers will be generated.
            endless_loop = true;
            endless_loop_well = j;
            break;
        end


    end

    if reduce_init_rate
        break;
    end



end

% Find final nonzero element in tubing (pWh)
if  endless_loop == false;
for i = 1:nWells
    if pTubing(end, i)==0
        X = find (pTubing(:, i) == 0);
        pTubing_end(i) = pTubing( X(1)-1, i);
    else
        pTubing_end(i) = pTubing(end, i);
    end


end
```

```
%% Pressure drop in riser and flowline, from Sep to PLEM.
n_steps_f = (l_riser+l_flowline)/step_length;

pRiser_Flow = zeros(n_steps_f+1, 1);
pRiser_Flow (end) = pSep;
T = TR;  % Should be made into a function.

for i = (n_steps_f):-1:1



    [ql, qg, dens_l, dens_g, visc_l, visc_g] =
local_properties(pRiser_Flow(i+1), T, sum(qo_sc), sum(qw_sc), sum(qg_sc),
sum(qgl_sc)); % Local fluid rates and properties

    if i<=l_flowline/step_length  % I.e. is the calculation in the flowline
or in the riser
        theta = asind(h_flowline/l_flowline);
    else
        theta = asind(h_riser/l_riser);
    end
    dpds = dpBeggs_Brill(ql, qg, dens_l, dens_g, visc_l, visc_g, theta,
step_length, ID_f);   % [Pa]

    pRiser_Flow(i) = pRiser_Flow(i+1) + dpds/bara;
end



n_steps_m = zeros(1, nManifold);  % Steps from PLEM to Manifold
for j = 1:nManifold
    n_steps_m(j) = l_PLEM_WH(j)/step_length;   % For tubing
end
pPLEM_WH = zeros(  max(n_steps_m)+1, nManifold);
pPLEM_WH(end, :) = pRiser_Flow(1, :);

for r = 1:nManifold     % "r" is a new variabel so not to disturb "j" from
the tubing-section.

    % To keep track of which wells are calculated:
    temp_10=0;
    for y = 1:r
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end

    for i = ( max(n_steps_m) ):-1: ( max(n_steps_m) - n_steps_m(r)+1    )




        [ql, qg, dens_l, dens_g, visc_l, visc_g] =
local_properties(pPLEM_WH(i+1, r), T, sum(qo_sc(temp_9+1:temp_10)),
sum(qw_sc(temp_9+1:temp_10)), sum(qg_sc(temp_9+1:temp_10)),
sum(qgl_sc(temp_9+1:temp_10))); % Local fluid rates and properties

        theta = asind(h_PLEM_WH(r)/l_PLEM_WH(r));
```

```
        dpds = dpBeggs_Brill(ql, qg, dens_l, dens_g, visc_l, visc_g, theta,
step_length, ID_m(r));   % [Pa]

        pPLEM_WH(i, r) = pPLEM_WH(i+1, r) + dpds/bara;
    end

end

for i = 1:nManifold
    if pPLEM_WH(1, i)==0
        X = find (pPLEM_WH(:, i) == 0);
        pPLEM_WH_end(1, i) = pPLEM_WH( X(end)+1, i);
    else
        pPLEM_WH_end(1, i) = pPLEM_WH(1, i);
    end
end

for i = 1:nManifold
    temp_10=0;
    for y = 1:i
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end

    dp_choke(1, temp_9+1:temp_10) = pTubing_end (1, temp_9+1:temp_10) -
pPLEM_WH_end(1, i);
end
end
if z<=nWells
%         if (min(dp_choke(active_wells))<-25)
%             min_temp = find(dp_choke <= min(dp_choke)+0.1);
%             if
abs(sum(abs(dp_choke(active_wells)))+sum(dp_choke(min_temp)))<(1*nWells/9)
% Proper scaling term?
%                 endless_loop = true;
%             end
%         end
    if isnan(sum(dp_choke)) || endless_loop % Imaginary numbers due to high
oil rate
         endless_loop = false;
%         if first_iteration
%             qo_sc_init = qo_sc;
%         end
        if qgl_sc(endless_loop_well)==0
            %qgl_sc(endless_loop_well)=qgl_sc_max*0.95;
            increased_init_gl = [increased_init_gl, endless_loop_well];
        else
            qo_sc_init(endless_loop_well) =
qo_sc_init(endless_loop_well)*0.9;
        end
        qgl_sc(:) = 0;
        qgl_sc(increased_init_gl) = qgl_sc_max*0.95;
        qo_sc = qo_sc_init;



        [pWf] = IPR_multi(qo_sc, pBP_0, pR);
        qw_sc = qo_sc.*wc;
        qg_sc = qo_sc.*GOR_0;
```

```
        %qgl_sc = zeros(1, nWells);
        active_wells = 1:nWells;
        %fprintf('Error: Too high rates in some of the wells result in NaN
dp_choke or an endless loop. All rates reduced by 1%%. \n');
        first_iteration = true;
        real_dp_choke = false;
        break;
    else
        dp_choke_mat(z, :) = dp_choke;
    end
else
   % disp(dp_choke);
end


if first_iteration
    re_original_rate = false;
    qo_sc_old = qo_sc; % = Initial rates
    qo_sc_init = qo_sc;
    qgl_sc_old = qgl_sc; % = 0
    if dp_choke>0
        break;  % I.e. all elements are positive.
    elseif dp_choke<0  % I.e. all elements are negative.
       % delta = ones(1, nWells)*qgl_sc_max;
        for i =1:nWells
            if qgl_sc_old(i)>0
                delta(i)=qgl_sc_max*0.05;
            else
                delta(i) = qgl_sc_max;
            end
            reduced_rate(i) = false;
            inactive_wells (i) = false;
            factor(i) = 1;
            original_rate (i) = false;
        end
    else % Some elements are negative, some positive.
        active_wells = [];
        for i = 1:nWells
            reduced_rate(i) = false;
            inactive_wells (i) = false;
            factor(i) = 1;
            original_rate (i) = false;
            if dp_choke(i)>0
                delta(i) = 0;
                inactive_wells (i) = true;
            else
                active_wells = [active_wells i];
                if qgl_sc_old(i)>0
                    delta(i)=qgl_sc_max*0.05;
                else
                    delta(i) = qgl_sc_max;
                end
            end
        end
    end
    qgl_sc(active_wells(1)) = delta(active_wells(1));
    dp_choke_old = dp_choke;
    first_iteration = false;
    break;
else
```

```
    if z<max(active_wells)
        qgl_sc = qgl_sc_old;
        qo_sc = qo_sc_old;
        temp = find(active_wells(:) == z); % Finds the active well
placement in "active_wells".
        if reduced_rate(active_wells(temp+1))
            qo_sc(active_wells(temp+1)) =
qo_sc_old(active_wells(temp+1))+delta(active_wells(temp+1));
        else
            qgl_sc(active_wells(temp+1)) =
qgl_sc_old(active_wells(temp+1))+delta(active_wells(temp+1));
        end

        for i = active_wells
            [pWf] = IPR_multi(qo_sc, pBP_0, pR);
            qw_sc(i) = qo_sc(i).*wc(i);
            qg_sc(i) = qo_sc(i).*GOR_0;
        end



    elseif z==max(active_wells)

        for i = active_wells
            if reduced_rate(i)
                qo_sc(i) = qo_sc_old(i) + delta(i);
            else
                qgl_sc(i) = qgl_sc_old(i) + delta(i);
            end
        end

        for i = active_wells
            [pWf] = IPR_multi(qo_sc, pBP_0, pR);
            qw_sc(i) = qo_sc(i).*wc(i);
            qg_sc(i) = qo_sc(i).*GOR_0;
        end



    else
    % Create the Jacobian:

    if (abs(dp_choke(active_wells))<0.1) % Ending criteria
        if (dp_choke(1, :)+0.1>0)
            if real_dp_choke
                first_iteration = true;
                real_dp_choke = true;
                break;
            end
        end
    end

    Jacobian = zeros(length(active_wells));
    count_i = 0;
    count_j = 0;
    for i = active_wells
        count_i = count_i+1;
       for j = active_wells
           count_j = count_j+1;
```

```
            Jacobian (count_i, count_j) = (dp_choke_mat(j, i)-
dp_choke_old(i))/delta(j);
        end
         count_j = 0;
    end

    % Calculate the new GL and oil -rates:
    qgl_sc_old = qgl_sc;
    qo_sc_old = qo_sc;
    qgl_sc = qgl_sc';
    qo_sc = qo_sc';


    dp_choke = dp_choke';
    now_active = [];
    dp_choke_active = [];

    for i = active_wells
        if reduced_rate(i)
            now_active = [now_active; qo_sc(i)];
        else
            now_active = [now_active; qgl_sc(i)];
        end
        dp_choke_active = [dp_choke_active; dp_choke(i)];
    end
    next_active = now_active - inv(Jacobian)*dp_choke_active;

    count = 0;
    for i = active_wells
        count = count+1;
        if reduced_rate(i)
%             if next_active(count)>qo_sc_min
              qo_sc(i) = next_active(count);
%             else
%                 qo_sc(i) = qo_sc(count)*0.9;
%             end
        else
            qgl_sc(i) = next_active(count);
        end
    end

    for i = active_wells
        if qgl_sc(i)<0 && dp_choke(i)<0  % This may happen for systems with
very large rates. If the previous dp_choke was negative, then the new GL-
rate should be positive.
            qgl_sc(i) = -qgl_sc(i);
        end
    end

    temp = active_wells;
    active_wells = [];
    for i = temp % Deactivate wells
        if qgl_sc(i)<=0  % Set this rate equal to zero, and exclude the
well from further calculations.
            qgl_sc(i) = 0;
            qo_sc(i) = qo_sc_init(i);
            qgl_sc_old(i) = 0;
            inactive_wells(i) = true;
         else
             active_wells = [active_wells i];
```

```
        end
    end
    qgl_sc = qgl_sc';
    qo_sc = qo_sc';

    active_wells =  [];
    for i=1:nWells
        if inactive_wells(i) && dp_choke(i)<0  % Re-activate the well
            qgl_sc(i) = qgl_sc_max;
            qo_sc(i) = qo_sc_init(i);
            active_wells = [active_wells i];
            inactive_wells(i) = false;
        elseif inactive_wells(i)==false
            active_wells = [active_wells i];
        end
    end
    if re_original_rate  % Well started with original (not reduced) oil
rate.
        for i = active_wells
            if original_rate(i)
                qgl_sc(i) = qgl_sc_max*0.5;
%                 qo_sc(i) = qo_sc_init(i);
                original_rate(i) = false;
%                 reduced_rate(i) = false;
            end
        end
        re_original_rate = false;
    end



    for i = active_wells
        if qgl_sc(i) > qgl_sc_max
            qgl_sc(i) = qgl_sc_max;
            if qgl_sc_old(i) ==qgl_sc_max
                delta(i) = qo_sc(i)*0.9-qo_sc(i);
                qo_sc(i) = qo_sc(i)*0.9;
                reduced_rate(i) = true;
            else
                delta(i) = qgl_sc(i) - qgl_sc_old(i);

            end
        else

            if reduced_rate(i)
                % (Below) Should not be qo_sc(i)<qo_sc_old(i) because then
                % it's impossible for the solver to recalculate within the
                % correct interval.

                if qo_sc(i)<qo_sc_init(i)*(0.9^factor(i))  % The rate
should be within initial or minimum rates.
                      factor (i) = factor(i)+1;

                        qo_sc(i) = qo_sc_init(i)*(0.9^factor(i));
                elseif qo_sc(i)>qo_sc_init(i) % The solver is searching for
GL<GL_max
                        qo_sc(i) = qo_sc_init(i);
                        reduced_rate(i) = false;
                        original_rate (i) = true;
                        re_original_rate = true;
                end
```

```
                delta(i) = qo_sc(i) - qo_sc_old(i);
            else
                delta(i) = qgl_sc(i) - qgl_sc_old(i);
            end

        end
    end

    qo_sc = qo_sc_old;  % Starting conditions for next iteration:
    qgl_sc = qgl_sc_old;
    if reduced_rate(active_wells(1))
        qo_sc(active_wells(1)) = qo_sc(active_wells(1)) +
delta(active_wells(1));
    else
        qgl_sc(active_wells(1)) = qgl_sc(active_wells(1)) +
delta(active_wells(1));
    end



        for i = active_wells
            [pWf] = IPR_multi(qo_sc, pBP_0, pR);
            qw_sc(i) = qo_sc(i).*wc(i);
            qg_sc(i) = qo_sc(i).*GOR_0;
        end


    dp_choke = dp_choke';
    dp_choke_old = dp_choke;









    disp(dp_choke);
    Jacobian = [];
    end
end


end

if first_iteration && real_dp_choke   % I.e. first dp_choke>0;
    break;
end
real_dp_choke = true;

end




disp(dp_choke);
```

```
%% Tidy-up section

% Pressures in the pipes:
p = zeros( (length(pTubing(:, 1))+length(pPLEM_WH(:,
1))+length(pRiser_Flow(:, 1)))-1, nWells);
p(1:length(pTubing), :) = pTubing;

for i = 1:nManifold
    temp_10=0;
    for y = 1:i
        temp_9 = temp_10;
        temp_10 = temp_10+wells_in_manifold(y);
    end
    for a = temp_9+1:temp_10
        p( (length(pTubing(:, 1))+1):(length(pTubing(:,
1))+length(pPLEM_WH(:, 1))) , a) = pPLEM_WH(:, i);
    end

end

for j =1:nWells
p ( (length(pTubing(:, 1))+length(pPLEM_WH(:, 1))):(length(pTubing(:,
1))+length(pPLEM_WH(:, 1))+length(pRiser_Flow(:, 1)))-1, j) = pRiser_Flow;
end

% GL rates:
for j =1:nWells
    temp = find(qgl_sc(:, j) ~= 0);
    if isempty(temp)
        qgl_sc_res(1, j) = 0;
    else
        qgl_sc_res(1, j) = qgl_sc(temp(end), j);
    end
end
qgl_sc_res = qgl_sc_res*day;

% Oil rates:
size_oil = size(qo_sc);
if size_oil(1)>1 % I.e. at least one well is calculated with reduced oil
rate.
    qo_sc_res = zeros(1, nWells);
    for k = 1:nWells
        nonzero_value = find(qo_sc(:, k) ~=0);
        correct_value = nonzero_value(end);
        qo_sc_res(k) = qo_sc(correct_value, k)*day;
    end
else
    qo_sc_res = qo_sc*day;   % I.e. Original oil rates count.
```

```
end

fprintf('Calculated oil rates [Sm3/D]: \n\n');
format short g;
disp(qo_sc_res);
fprintf('Calculated GL rates [Sm3/D]: \n\n');
disp(qgl_sc_res);

end
```

“multi_dTdx_pipe.m”

```
function T2 = multi_dTdx_pipe(T1, p, theta, rho_m, rho_ns, ge, Cp, holdup,
dpds, dvds, A, gamma_g, vm, T_amb)

global  step_length
%% Calculation of the Joule Thomson Coefficient

z1 = z_Standing(p, T1, gamma_g);
z2 = z_Standing(p, T1+0.1, gamma_g);
dzdT = (z2-z1)/0.1;

eta = -1/(Cp*rho_ns)* ((1-holdup)*(-T1/z1*dzdT) + holdup);


%% Calculation of the dimensionless parameter (Ramsey 1962)

phi = (rho_m *eta*Cp*dpds - rho_m*9.81*sind(theta)-rho_m*vm*dvds)/dpds;

%% Calculation of the Temperature

T2 = T_amb - ge*step_length*sind(theta) + (T1-T_amb)*exp(-step_length/A)
...
    + ge*A*sind(theta)*(1-exp(-step_length/A)) + dpds*phi*A*(1-exp(-
step_length/A))/(Cp*rho_m);
if T2<T_amb
    T2 = T_amb; % Valid approximation?
else

end

end
```

“Utocalc.m”

```
function res = Utocalc(hc, hi, rti, rto, kt, rwb, rco, rci, kcas, kcem)
% 'Function to calculate the overall heat transfer coefficient,
Btu/hr*ft^2*°F
% 'hc, convective heat transfer coefficient for annulus fluid  Btu/°F hr
ft^2
% 'rto, tubing outside radious, ft
% 'rwb, Outside radius of wellbore, ft
% 'rco, outside radius of casing, ft
% 'kcem, cement conductivity, Btu/hr ft °F

hi = hi*0.176110184;
res = (  rti/(rti*hi) + rti*log(rto/rti)/kt + rti /(rto* hc) +
rti*log(rco/rci)/kcas + (rti * log(rwb / rco) / kcem)   ) ^ -1;

end
```

“Tempfluidcalc.m”

```
function res = Tempfluidcalc(Tei, zbh, z, g, teta, cp, fi, gt, Tfbh, Tebh,
A)
% 'Tei, Undisturbed formation temperature at any given depth, °F
% 'zbh, Total well depth from surface, ft
% 'z, variable well depth from surface, ft
% 'g gravitational acceleration, ft/sec^2
% 'teta, Pipe inclination angle form horizontal, degrees
% 'gc Conversion factor, lbm ft/lbf sec^2
gc = 32.2;
% 'J mechanical equivalent of heat ft-lbf/Btu
J = 778;
% 'cp fluid specific heat capacity, Btu/lbm °F
% 'fi, correction factor
% 'gt, geothermal temperature gradient, °F/ft
% 'Tfbh, temperature of the fluid at the bottomhole, °F
% 'Temperature of the formation at the bottomhole, °F
% 'inverse relaxation distance, ft
res = Tei + A * (1 - exp(-(zbh - z) / A)) * ((-g * abs(sind(teta)) / (gc *
J * cp)) + fi + (gt * abs(sind(teta)))) + exp(-(zbh - z) / A) * (Tfbh -
Tebh);
end
```

“TempD.m”

```
function res = TempD(Td)
%'Function to calculate the dimensionless temperature
    if Td >= 0.000000001 && Td <= 1.5
        res = 1.1281 * (Td ^ 0.5) * (1 - 0.3 * (Td ^ 0.5));
    end
    if Td > 1.5
        res = (0.4063 + 0.5 * log(Td)) * (1 + (0.6 / Td));
    end
end
```

“TempD.m”

```
function res = Td(alfa, T, rwb)
% 'Function to calculate the dimensionless time
% 'alfa, Thermal difussivity of earth,  [ft^2/hr]
% 'rwb, well bore diameter [ft]
% 't, Production time, [hr]
    res = (alfa * T) / (rwb ^ 2);
end
```

“Tcicalc.m”

```
function res = Tcicalc(Tf, Td, ke, rto, Uto, Te, rwb, rco, kcem, Tto)
% 'Function to clear the  casing inner wall temperature combining heat
transfer equations through the casing and cement with the equation of
steady state from the wellbore
% 'Tf Fluid temperature, °F
% 'Td, dimensionless temperature
% 'ke, Earth formation conductivity, Btu/hr ft °F
% 'rto, tubing outside radious, ft
% 'Uto, Overall heat transfer coefficient, Btu/hr ft^2 °F
% 'Te, Formation temperature at any given depth, °F
% 'rwb, Outside radius of wellbore, ft
% 'rco, outside radius of casing, ft
% 'kcem, cement conductivity, Btu/hr ft °F[\
% 'Tto, °F

Twbcalc = ((Tf * Td) + (ke * Te / (rto * Uto))) / (Td + (ke / (rto *
Uto)));
res = Twbcalc + ((Tto - Twbcalc) * rto * Uto * log(rwb / rco) / kcem);
end
```

“Npr.m”

```
function res = Npr(Cpa, mua, ka)
% 'Prandtl Number for annulus
% 'Cpa Annular fluid specific heat capacity, Btu/lbm °F
% 'mua,Annular fluid viscosity, cp
% 'ka, Annular fluid conductivity, Btu/hr ft °F
res = Cpa * mua / ka;
end
```

“NGr.m”

```
function res = NGr(rci, rto, roa, beta, Tto, Tci, mua)
% 'Calculates the grashof number for free convection heat transfer
calculations in the annulus
% 'rci,casing inside radius, ft
% 'rto, tubing outside radious, ft
% 'g gravitational acceleration, ft/sec^2
% 'roa, Annular fluid density, lbm/ft^3
% 'beta, (annulus temperature)^-1, 1/°R
% 'Tto, Tubing outside temperature, °F
% 'Tci, Casing inner wall temperature, °F
% 'mua, Annular fluid viscosity, cp

g = 32.185 ;%'[ft/s^2]

if Tto<=Tci
    res = ((rto - rci) ^ 3) * g * (3600 ^ 2) * (roa ^ 2) * beta * (Tci -
Tto) / (mua ^ 2);
else
    res = ((rci - rto) ^ 3) * g * (3600 ^ 2) * (roa ^ 2) * beta * (Tto -
Tci) / (mua ^ 2);
end

end
```

“multi_dTdx_tubing_simple.m”

```
function [res, HTC] = multi_dTdx_tubing_simple(T1, p, theta, rho_m, rho_ns,
ge, Cp, holdup, dpds, dvds, Wt, gamma_g, vm, T_amb)

global ro_t k_steel step_length ID_t

%% Calculation of the Joule Thomson Coefficient
% T1 is inlet fluid temperature.
z1 = z_Standing(p, T1, gamma_g);
z2 = z_Standing(p, T1+0.1, gamma_g);
dzdT = (z2-z1)/0.1;
eta = -1/(Cp*rho_ns)* ((1-holdup)*(-T1/z1*dzdT) + holdup);

% Calculation of the dimensionless parameter (Ramsey 1962)

fi = (rho_m *eta*Cp*dpds - rho_m*9.81*sind(theta)-rho_m*vm*dvds)/dpds;

%% Calculation of the Heat Transfer
Layerinfoeach (1) = ID_t/2 * 3.2808399; % Tubing inner radius[Feet]; %
Default
Layerinfoeach (2) = ro_t*3.2808399; % Tubing outer radius[Feet]
Layerinfoeach (3) = k_steel/1.7; % Tubing wall conductivity [Btu/hr ft °F]
Layerinfoeach (4) = Layerinfoeach (2)+4/12; % Outer radius of Annulus
(Casing inner diameter) (4") [Feet]
Layerinfoeach (5) = 0.383; % Annulus conductivity [Btu/hr ft °F]
Layerinfoeach (6) = 1; % Annular fluid Heat Capacity [Btu/lbm °F]
Layerinfoeach (7) = 1; % Annular fluid Viscosity [cp]
Layerinfoeach (8) = 62.373; % Annular fluid density [lbm/ft^3]
```

```
Layerinfoeach (9) = Layerinfoeach (4) + 0.8/12; % Casing outer radius
(0.8") [Feet]
Layerinfoeach (10) = 21; % Casing conductivity [Btu/hr ft °F]
Layerinfoeach (11) = Layerinfoeach (9) + 2/12; % Cement outer radius (r
well bore) (2") [Feet]
Layerinfoeach (12) = 4.021; % Cement conductivity [Btu/hr ft °F]

kearth = 0.83; % [Btu/hr ft °F]
Difearth = 0.04; % [ft^2/hr]
prodtime = 164.0; % [Hr]
td1each = TempD(Td(Difearth, prodtime, Layerinfoeach(11)));

HTC = HTC_calc(Layerinfoeach (1), Layerinfoeach (2), Layerinfoeach (4), ...
        Layerinfoeach (5), Layerinfoeach (11), Layerinfoeach (9),
Layerinfoeach (12) );


%% Calculation of the Temperature
     A = Acalc(Cp, Wt, kearth, Layerinfoeach(1, 1), HTC, td1each);

    res = T_amb - ge*step_length*sind(theta) + (T1-T_amb)*exp(-
step_length/A) ...
        + ge*A*sind(theta)*(1-exp(-step_length/A)) + dpds*fi*A*(1-exp(-
step_length/A))/(Cp*rho_m);

end
```

“multi_dTdx_tubing_simple.m”

```
  function [res, Uglobalcalc] = multi_dTdx_tubing(T1, p, theta, rho_m,
rho_ns, ge, Cp, holdup, dpds, dvds, Wt, gamma_g, vm, T_amb, hi)

global ro_t k_steel step_length ID_t

%% Calculation of the Joule Thomson Coefficient
% T1 is inlet fluid temperature.
z1 = z_Standing(p, T1, gamma_g);
z2 = z_Standing(p, T1+0.1, gamma_g);
dzdT = (z2-z1)/0.1;
Temperearth = (T_amb - ge*step_length*sind(theta))*9/5-459.67; % Earth temp
at outlet. T_amb is at inlet.
eta = -1/(Cp*rho_ns)* ((1-holdup)*(-T1/z1*dzdT) + holdup);


% Calculation of the dimensionless parameter (Ramsey 1962)


fi = (rho_m *eta*Cp*dpds - rho_m*9.81*sind(theta)-rho_m*vm*dvds)/dpds;

%% Calculation of the Heat Transfer
Layerinfoeach (1) = ID_t/2 * 3.2808399; % Tubing inner radius[Feet]; %
Default
Layerinfoeach (2) = ro_t*3.2808399; % Tubing outer radius[Feet]
Layerinfoeach (3) = k_steel/1.7; % Tubing wall conductivity [Btu/hr ft °F]
Layerinfoeach (4) = Layerinfoeach (2)+4/12; % Outer radius of Annulus (4")
[Feet]
```

```
Layerinfoeach (5) = 0.383; % Annulus conductivity [Btu/hr ft °F]
Layerinfoeach (6) = 1; % Annular fluid Heat Capacity [Btu/lbm °F]
Layerinfoeach (7) = 1; % Annular fluid Viscosity [cp]
Layerinfoeach (8) = 62.373; % Annular fluid density [lbm/ft^3]
Layerinfoeach (9) = Layerinfoeach (4) + 0.8/12; % Casing outer radius
(0.8") [Feet]
Layerinfoeach (10) = 21; % Casing conductivity [Btu/hr ft °F]
Layerinfoeach (11) = Layerinfoeach (9) + 2/12; % Cement outer radius (2")
[Feet]
Layerinfoeach (12) = 4.021; % Cement conductivity [Btu/hr ft °F]


kearth = 0.83; % [Btu/hr ft °F]
Difearth = 0.04; % [ft^2/hr]
prodtime = 164.0; % [Hr]
Nprcalc = Npr(Layerinfoeach(1, 6), Layerinfoeach(1, 7), Layerinfoeach(1,
5));
td1each = TempD(Td(Difearth, prodtime, Layerinfoeach(11)));

Temp = T1*9/5-459.67; % From Kelvin to Fahrenheit
Uglobal = 8;
 Epsilon = 8;
 while Epsilon >= 0.0001
        Tci = Tcicalc(Temp, td1each, kearth, Layerinfoeach(1, 1), Uglobal,
Temperearth, Layerinfoeach(1, 11), Layerinfoeach(1, 9), Layerinfoeach(1,
12), Temp);
%         if Tci <= Temp
            betaa = (1 / (Annulartemp(Tci, Temp) + 459.67));
            NGrcalc = NGr(Layerinfoeach(1, 4), Layerinfoeach(1, 2),
Layerinfoeach(1, 8), betaa, Temp, Tci, Layerinfoeach(1, 7));
            hc = hccalc(NGrcalc, Nprcalc, Layerinfoeach(1, 5),
Layerinfoeach(1, 2), Layerinfoeach(1, 4));
            Uglobalcalc = Utocalc(hc, hi, Layerinfoeach(1, 1),
Layerinfoeach(1, 2), Layerinfoeach(1, 3), Layerinfoeach(1, 11),
Layerinfoeach(1, 9), Layerinfoeach(1, 4), Layerinfoeach(1, 10),
Layerinfoeach(1, 12));
            Uglobal = Uglobalcalc;

%% Calculation of the Temperature
     A = Acalc(Cp, Wt, kearth, Layerinfoeach(1, 1), Uglobalcalc, td1each);

    TempTemp = T_amb - ge*step_length*sind(theta) + (T1-T_amb)*exp(-
step_length/A) ...
        + ge*A*sind(theta)*(1-exp(-step_length/A)) + dpds*fi*A*(1-exp(-
step_length/A))/(Cp*rho_m);

     Epsilon = abs(TempTemp*9/5-459.67 - Temp);
     Temp = TempTemp*9/5-459.67; % From Kelvin to Fahrenheit;
 end
% Uglobalcalc = 2;
%      A = Acalc(Cp, Wt, kearth, Layerinfoeach(1, 1), Uglobalcalc,
td1each);
%
%      TempTemp = T_amb - ge*step_length*sind(theta) + (T1-T_amb)*exp(-
step_length/A) ...
%         + ge*A*sind(theta)*(1-exp(-step_length/A)) + dpds*fi*A*(1-exp(-
step_length/A))/(Cp*rho_m);
res = TempTemp;

end
```

“HTC_calc.m”

```
function HTC = HTC_calc(rti, rto, rci, kan, rwb, rco, kcem)

% rti = Tubing inner radius [feet]
% rto = Tubing outer radius [feet]
% rci = Casing inner radius [feet]
% rwb = WellBore inner radius [feet]
% rco = Casing outer radius [feet]
% kan = Annular conductivity [Btu/hr ft °F]
% kcem = Cement conductivity [Btu/hr ft °F]
% HTC = Overall Heat Transfer Coeff [Btu/hr ft^2 °F]

Inverted = rti* ( log(rci/rto)/kan + log(rwb/rco)/kcem);
HTC = Inverted^-1;
end
```

“Fc.m”

```
function res = Fc(Pwh, Wt, GLR, API, gammag, gt)
% 'Term that combines the Joule-Thompson effect and the kinetic energy
% 'Pwh Well head pressure,psig
% 'Wt, total mass flow rate, lbm/s
% 'GLR gas liquid ratio, scf/STB
% 'API, Oil gravity °API
% 'gammag, gas specific gravity
% 'gt Geothermal gradient °F/ft
    res = -0.002978 + (0.000001006 * Pwh) + (0.0001906 * Wt) - (0.000001047
* GLR) + (0.00003229 * API) + (0.00409 * gammag) - (0.3551 * gt);
end
```

“Annulartemp.m”

```
function res = Annulartemp(Tf, Tci)
% 'Calculates the Annulus temperature perfoming an average between inner
casing temperature and fluid temperature
% 'Tf, fluid temperature, °F
% 'Tci, Casing inner wall temperature, °F
res = (Tf + Tci) / 2;

end
```

“Acalc.m”

```
function res = Acalc(cp, Wt, ke, rto, Uto, Td)
% 'Inverse relaxation distance, [ft]
% 'cp fluid specific heat capacity, Btu/lbm °F
% 'Wt= Mass flow rate lbm/s
% 'ke, Earth formation conductivity, Btu/hr ft °F
% 'rto, tubing outside radius, ft
```

```
% 'Uto, Overall heat transfer coefficient, Btu/hr ft^2 °F
% 'Td, dimensionless temperature

cp = cp*0.000238846; % Convert from J/Kg C
Wt = Wt*2.20462262; % Convert from Kg

res = (cp * Wt * 3600 / (2 * 3.141592)) * (ke + (rto * Uto * Td)) / (rto *
Uto * ke);

res = res/3.2808399;  % From feet to m
end
```

“z_Standing.m”

```
function [z, Ppr, Tpr] = z_Standing(p, T, gamma_g)
T = T-273.15; % [C]
T = 9 / 5 * T + 32;  % [F]
p = p*14.5;    % [psi]

% Metric system

% Sutton, R.P. 1985.
Tpc = 169.2 + 349.5 * gamma_g - 74 * gamma_g ^ 2;
Ppc = 756.8 - 131.07 * gamma_g - 3.6 * gamma_g ^ 2;

Tpr = (T + 460) / Tpc;
Ppr = p / Ppc;


T = 1 / Tpr;
a = 0.06125 * T * exp(-1.2 * (1 - T) ^ 2);
y = 0.001;
    for i = 1:100
    fy = -a * Ppr + (y + y ^ 2 + y ^ 3 - y ^ 4) / (1 - y) ^ 3 - (14.76 * T
- 9.76 * T ^ 2 + 4.58 * T ^ 3) * y ^ 2 + (90.7 * T - 242.2 * T ^ 2 + 42.4 *
T ^ 3) * y ^ (2.18 + 2.82 * T);
    dfY = (1 + 4 * y + 4 * y ^ 2 - 4 * y ^ 3 + y ^ 4) / (1 - y) ^ 4 -
(29.52 * T - 19.52 * T ^ 2 + 9.16 * T ^ 3) * y + (2.18 + 2.82 * T) * (90.7
* T - 242.2 * T ^ 2 + 42.4 * T ^ 3) * y ^ (1.18 + 2.82 * T);
    if abs(fy)<0.0001
        break;
    end
    y = y - fy / dfY;   % Newton-Rhapson formula
    end

z = a * Ppr / y;


end
```

“STwg.m”

```
%   Water-gas interfacial tension in dyn/cm

function res = STwg(T)
%Temperature, °F

T = T*9/5-459.67; % From K to F

    res = 72.75 * (1 - 0.001171 * (T - 60) + 0.000001121 * (T - 60) ^ 2);

end
```

“STog.m”

```
%   Oil-Gas interfacial tension in dyn/cm

function res = STog(deno, deng)
%Density of the oil,kg/m^3
%Density of the gas,kg/m^3

    res = (567 * (deno - deng) * 0.001 / 213) ^ 4;

end
```

“solution_GOR.m”

```
function Rs = solution_GOR(p, T, gamma_g, pBP)

% Standing correlation (1947)

global gamma_o


if p>=pBP
    p = pBP;
end


Rs = 0.0059 * gamma_g *10^(2.14/gamma_o)*10^(-0.00198*T) * (0.797*p +
1.4)^1.205;

end
```

“oil_visc_standing.m”

```
function visc_o = oil_visc_standing(p, T, Rs, pBP)

global API_o

% Standing correlation, SI units [bar, K, cp].

visc_od = dead_visc_oil(T, API_o);    % Dead oil at T [cp]


a = 10^(Rs *(6.9*10^(-6)*Rs-4.2*10^(-3)));
b = 0.68 * 10^(-4.84*10^(-4)*Rs)+0.25*10^(-6.18*10^(-3)*Rs)+0.062*10^(-
2.1*10^(-7)*Rs);
visc_o = a*(visc_od)^b;

if p>pBP
    visc_o = visc_o+10^(-3)*(0.35*visc_o^1.6 + 0.55*visc_o^0.56)*(p-pBP);
end

end
```

“local_properties.m”

```
function [ql, qg, dens_l, dens_g, visc_l, visc_g, SIG, gamma_g, Cp] =
local_properties(p, T, qo_sc, qw_sc, qg_sc, qgl_sc)

global gamma_g_0 gamma_gl_0 visc_g_sc_0 GOR_0 R dens_o_sc dens_w_sc p_sc
T_sc M_air bara

if qgl_sc~=0
    qg_tot_sc = qg_sc + qgl_sc;
    GOR = qg_tot_sc/qo_sc;
    gamma_g = (qg_sc*gamma_g_0 + qgl_sc*gamma_gl_0)/(qg_tot_sc);
    pBP = bubble_point(GOR, T, gamma_g);
    visc_g_sc = dead_visc_gas(gamma_g);
else  % No GL
    qg_tot_sc = qg_sc;
    GOR = GOR_0;
    gamma_g = gamma_g_0;
    pBP = bubble_point(GOR, T, gamma_g);
    visc_g_sc = visc_g_sc_0;
end
Rs = solution_GOR(p, T, gamma_g, pBP);
[z, Ppr, Tpr] = z_Standing(p, T, gamma_g);
dens_g_sc = gamma_g*M_air*p_sc*bara/(R*T_sc);
[Bo, Bw, Bg] = FVF (gamma_g, Rs, p, T, z, pBP);

qg =(qg_tot_sc-Rs*qo_sc)*Bg;
if qg<0
    qg = 0;
end
qo = qo_sc*Bo + Rs * qo_sc*Bg;
qw = qw_sc*Bw;
ql = qo+qw;
```

```
dens_o = (dens_o_sc+dens_g_sc*Rs)/Bo;
dens_g = dens_g_sc/Bg;
dens_w = dens_w_sc/Bw;
dens_l = (dens_w*qw + dens_o*qo)/ql;

gamma_l = (dens_o_sc*qo + dens_w_sc*qw)/(ql*dens_w_sc);

% [cp]
visc_o = oil_visc_standing(p, T, Rs, pBP);
%visc_w = water_visc2(T);
if T<273.15
    visc_w = visc_o;
else
    visc_w = XSteam('my_pT', p, T-273.15)*1000;
end
visc_g = gas_visc(Ppr, Tpr, visc_g_sc);

% TODO: Find better correlation for mixture viscosity.
% wc_cuttoff = 0.6;
% wc = (qw/ql);
% if wc > wc_cuttoff
%     expw = 3.089;
%     visc_l = visc_w*exp(expw*(1-wc));
% else
%     expo = 3.215;
%     visc_l = visc_o*exp(expo*(wc));
% end

% http://people.clarkson.edu/~wwilcox/Design/HYSYSpropSelect.pdf

if qo/ql>=0.5
    visc_l = visc_o*exp(3.6*(1-qo/ql));
elseif qo/ql<=0.33
    visc_l = (1+2.5*qo/ql*(visc_o+0.4*visc_w)/(visc_o+visc_w))*visc_w;
else
    visc_weighted1 = visc_o*exp(3.6*(1-qo/ql));
    visc_weighted2 =
(1+2.5*qo/ql*(visc_o+0.4*visc_w)/(visc_o+visc_w))*visc_w;
    visc_l = ((qo/ql-0.33)*visc_weighted1 + (0.5-
qo/ql)*visc_weighted2)/(0.5-0.33);
end

% Surface tensions:

fw = qw/qo;
% Gas surface tension(?)
SIG = (STog(dens_o, dens_g) * 0.001 * (fw + 1)) + (fw * 0.001 * STwg(T));

%% Calculation of Cp

[Cpo, Cpg] = HeatCapacity(gamma_g, T);
if T<273.15
    Cpw = Cpo;
else
    Cpw = XSteam('Cp_pT', p, T-273.15)*1000;
end
Cp = (dens_o*qo*Cpo + dens_w*qw*Cpw + dens_g*qg*Cpg)/(dens_l*ql+dens_g*qg);
end
```

“IPR_multi.m”

```
function [pWF] = IPR_multi(qo, pBP, pR)

% Note. This IPR is NOT optimal, and sometimes produce weird results.

qo = qo*24*3600;  % Change the sign to positive values
J = 40;   % [Sm3/D/Bar]
endless_loop_temp = false;
qo_max = J*(pR-pBP+pBP/(1.8));   %  [Sm3/D];
% while qo>0

        pWF = pR - qo./J;
        for i = 1:length(pWF)
%             if qo(i)>qo_max
%                 pWF(i) = -1;
%                 qo(i) = qo(i)*0.99;
%                 break;
%             end
        if pWF(i) <= pBP
            qb = J * (pR - pBP);  % Flowrate at pWF = pBP

            a = 0.8*(qb-qo_max)/(pBP^2);
            b = 0.2*(qb-qo_max)/pBP;
            c = qo_max-qo(i);
            pWF(i) = (-b - sqrt(b.^2-4.*a.*c))/(2*a);  % ABC-formula
        end
            if pWF(i)<0
                qo(i) = qo(i)*0.99;
                break;
            end
        end
%         if pWF>0
%             if endless_loop_temp
%                 endless_loop = true;
%             else
%                 endless_loop = false;
%             end
%             break; % Out of while loop and return variables.
%         else
%             endless_loop_temp = true;
%             fprintf('At least one of the wells have too high rates,
returning a negative pWF. The rate is reduced\n\n');
%             break;
%         end
% end
end
```

“HeatCapacity.m”

```
function [Cpo, Cpg] = HeatCapacity(SGg, T)

global Tbnoil Tbngas gamma_o
SGo = gamma_o;
T = T * 9/5 - 459.67; % Convert temperature from K to F

Kwoil = (Tbnoil ^ (1 / 3)) / SGo;
Kwgas = (Tbngas ^ (1 / 3)) / SGg;

Cpo = (((0.055 * Kwoil) + 0.35) * (0.6811 - (0.308 * SGo)) + (T * (0.000815
- 0.000306 * SGo))) *4186.8; % [joule/kilogram/K]
Cpg = (((4 - SGg) / 6450) * (T + 670) * ((0.12 * Kwgas) - 0.41)) *4186.8; %
[joule/kilogram/K]

% 1 Btu (IT)/pound/°F = 4186.800000009 joule/kilogram/K
end
```

“gas_visc.m”

```
function visc_g = gas_visc(Ppr, Tpr, visc_g_sc)

%  Dempsey (1965)

 a0 = - 2.46211820;
 a1 = 2.97054714;
 a2 = - 0.286264054;
 a3 = 8.05420522*10^(-3);
 a4 = 2.80860949;
 a5 = - 3.49803305;
 a6 = 0.360373020;
 a7 = - 1.044324*10^(-2);
 a8 = - 0.793385684;
 a9 = 1.39643306;
 a10 = - 0.149144925;
 a11 = 4.41015512 *10^(-3);
 a12 = 8.39387178 *10^(-2);
 a13 = - 0.186408848;
 a14 = 2.03367881*10^(-2);
 a15 = -6.09579263*10^(-4);

 b = a0 + a1*Ppr + a2*Ppr^2 + a3*Ppr^3 + Tpr*(a4 + a5*Ppr + a6*Ppr^2 +
a7*Ppr^3) ...
 + Tpr^2*(a8 + a9*Ppr + a10*Ppr^2 + a11*Ppr^3) + Tpr^3*(a12 + a13*Ppr +
a14*Ppr^2 + a15*Ppr^3);

visc_g = exp(b)*visc_g_sc/Tpr;

end
```

“FVF.m”

```
function [Bo, Bw, Bg] = FVF (gamma_g, Rs, p, T, z, pBP)

global gamma_o p_sc T_sc dens_w_sc

Bg = p_sc*T*z/(p*T_sc);


Bo = 0.9759+0.952*10^(-3)*((gamma_g/gamma_o)^0.5*Rs+0.401*T-103)^1.2;  %
Approximates Bo_pBP when Rs=GOR

if T<273.15
    Bw = 0.9759+0.952*10^(-3)*(0.401*T-103)^1.2;  % Ok Approximation?
else
    Bw = dens_w_sc/XSteam('rho_pT', p, T-273.15);
end

if p>pBP
    c = 10^(-4)*(2.81*Rs+3.10*T+171/gamma_o-118*gamma_g-1102)/p;    % Oil
compressibility, Vazquez and Beggs (1980)
    Bo = Bo*exp(-c*(p-pBP));
end
end
```

“dead_visc_oil.m”

```
function visc_od = dead_visc_oil(T0, API)
% Viscosity of dead oil, Glasø correlation (1980)
T = T0*9/5-459.67;    % Convert from Kelvin to Farenheit in this formula

visc_od = 3.141*10^10*T^(-3.444)*log10(API)^(10.313*log10(T)-36.447); %
[cp]

end
```

“dead_visc_gas.m”

```
function visc_g_sc = dead_visc_gas(gamma_g)

global T_sc

visc_g_sc = (3.0764*10^(-5) - 3.712*10^(-6)*gamma_g)*(T_sc-256) +
8.188*10^(-3)- 6.15*10^(-3)*log10(gamma_g); % Standing Correlation
end
```

“bubble_Uo.m”

```
%Bubble_rise_velocity
function res = bubble_Uo(RHOL, RHOg, SIG, beta)
% Calculation of bubble rise velocity for bubbly flow regime

% Uo = bubble rise velocity
    res = sin(beta) * 1.53 * ((9.81 * abs(RHOL - RHOg) * SIG) / RHOL ^ 2) ^
0.25;
end
```

“bubble_point.m”

```
function pBP = bubble_point(GOR, T, gamma_g)

% Standing correlation (1947), SI Units
global gamma_o


pBP = 1.255*(GOR/(0.0059*gamma_g*10^(2.14/gamma_o)*10^(-0.00198*T)))^0.83 -
1.76;


end
```

“bubble_H.m”

```
%Bubbly_flow_holdup
function res = bubble_H(UGS, ULS, Uo)
% Calculation of liquid holdup for bubbly flow

% Uo = bubble rise velocity
if abs(Uo) < 0.00001
    Uo = 0.00001;
end

    MM = Uo + UGS + ULS;
% Alfab = void fraction
    res = (MM - (MM ^ 2 - 4 * Uo * UGS) ^ 0.5) / (2 * Uo);


end
```

“dpBeggs_Brill.m”

```
function [dpds, dens_m, dens_ns, Hl] = dpBeggs_Brill(ql, qg, dens_l,
dens_g, visc_l, visc_g, theta, step_length, ID)

global g

v_sg = qg/(pi*(ID^2)/4);
v_sl = ql/(pi*(ID^2)/4);
v_m = v_sg + v_sl;

Cl = v_sl/v_m;  % No slip liquid holdup

%% Liquid holdup correlation here:
% Beggs and Brill.
L1 = 316*Cl^0.302;
L2 = 0.0009252*Cl^(-2.4684);
L3 = 0.1*Cl^(-1.4516);
L4 = 0.5*Cl^(-6.738);
Frm = v_m^2/(g*ID);
% sigma = 0.0012;   % Assumed liquid surface tension. Correlation should be
implemented instead.
% Nlv = 1.938*v_sl*(dens_l/sigma)^(1/4);   % Uncertain about the units that
should be used. Nlv is almost dimensionless, but not quite.
sigma = 50; % Dynes/cm
Nlv = 1.938*v_sl/0.3048*(dens_l*0.0624279606/sigma)^(1/4);  % Converted to
field units.
Hl = 1;

if (Cl<0.01 && Frm<L1) || (Cl>=0.01 && Frm<L2)
    % Segregated flow
    Hl_0 = (0.98*Cl^0.4846)/(Frm^0.0868);
    if theta>=0
        beta = (1-Cl)*log(0.011*Nlv^3.539*Cl^-3.768*Frm^-1.614);
    else
        beta = (1-Cl)*log(4.70*Nlv^0.1244*Cl^-0.3692*Frm^-0.5056);
    end
    if beta < 0
        beta =0;
    end

    B_theta = 1+beta*(sind(1.8*theta)-(sind(1.8*theta))^3/3);
    Hl = Hl_0 * B_theta;
elseif (Cl>=0.01 && Cl<0.4 && Frm>L3 && Frm<=L1) || (Cl>=0.4 && Frm>L3 &&
Frm<=L4)
    % Intermittent flow
    Hl_0 = (0.845*Cl^0.5351)/(Frm^0.0173);
    if theta>=0
        beta = (1-Cl)*log(2.96*Nlv^-0.4473*Cl^0.305*Frm^-0.0978);
    else
        beta = (1-Cl)*log(4.70*Nlv^0.1244*Cl^-0.3692*Frm^-0.5056);
    end
    if beta < 0
        beta =0;
    end

    B_theta = 1+beta*(sind(1.8*theta)-(sind(1.8*theta))^3/3);
    Hl = Hl_0 * B_theta;
elseif (Cl<0.4 && Frm>=L1) || (Cl>=0.4 && Frm>L4)
```

```
    % Distributed flow
    Hl_0 = (1.065*Cl^0.5824)/(Frm^0.609);

    if theta>=0
        beta = 0;
    else
        beta = (1-Cl)*log(4.70*Nlv^0.1244*Cl^-0.3692*Frm^-0.5056);
    end
    if beta < 0
        beta =0;
    end

    B_theta = 1+beta*(sind(1.8*theta)-(sind(1.8*theta))^3/3);
    Hl = Hl_0 * B_theta;
elseif (Cl>=0.001 && Frm>L2 && Frm<L3)
    % Transitional flow
    A = (L3 - Frm)/(L3 - L2);
    B = 1-A;

        % Segregated flow
    Hl_0_s = (0.98*Cl^0.4846)/(Frm^0.0868);
        % Intermittent flow
    Hl_0_i = (0.845*Cl^0.5351)/(Frm^0.0173);
    if theta>=0
        beta_s = (1-Cl)*log(0.011*Nlv^3.539*Cl^-3.768*Frm^-1.614);
        beta_i = (1-Cl)*log(2.96*Nlv^-0.4473*Cl^0.305*Frm^-0.0978);
    else
        beta_s = (1-Cl)*log(4.70*Nlv^0.1244*Cl^-0.3692*Frm^-0.5056);
        beta_i = beta_s;
    end
    if beta_s < 0
        beta_s =0;
    end
    if beta_i < 0
        beta_i =0;
    end

    B_theta_s = 1+beta_s*(sind(1.8*theta)-(sind(1.8*theta))^3/3);
    B_theta_i = 1+beta_i*(sind(1.8*theta)-(sind(1.8*theta))^3/3);
    Hl_s = Hl_0_s * B_theta_s;
    Hl_i = Hl_0_i * B_theta_i;

    Hl = A*Hl_s + B*Hl_i;
end

if Hl<Cl
    Hl=Cl;
end
if Hl>1
    Hl = 1;
end


%%


dens_m = dens_l*Hl + dens_g*(1-Hl);    % Slip included (true mixture
values)
%visc_m = visc_l*Hl + visc_g*(1-Hl);
```

```
dens_ns = dens_l*Cl + dens_g*(1-Cl);    % No slip
visc_ns = visc_l*Cl + visc_g*(1-Cl);

% Previously 1488
Re_ns = 1000 * (dens_ns*v_m*ID)/visc_ns;   % Viscosity in cp.
f_ns = (2*log10(Re_ns/(4.5223*log10(Re_ns)-3.8215)))^-2;   % No-slip
friction factor in smooth pipe
x = Cl/(Hl^2);

if x==0
    S = 0;
elseif x>1 && x<1.2
    S = log(2.2*x-1.2);
else
    S = log(x)/(-0.0523+3.182*log(x)-0.8725*(log(x))^2+0.01853*(log(x))^4);
end

f_tp = f_ns*exp(S);   % Two phase friction factor.

dp_fric = f_tp/2 * dens_ns * v_m^2*step_length/ID;
dp_grav = g*dens_m*step_length*sind(theta);

dpds = dp_fric + dp_grav;

% Accelleration may also be included. See Thiago's paper. Here the
% dimensions/units seem wrong, though.

end
```

“multi_dpdx.m”

```
function [DPDX, DENS, HOLDUP] = multi_dpdx(ql, qg, deno, deng, visco,
viscg, angle, d, intpt)
% d = Inner Diameter (ID)
% deno = dens_l
% visc_o = visc_l
% The variable names are changed here because the functions are written
% from the work of others.
angle = degtorad(angle);   % The angle is given by user in degrees.
visco = visco/1000; % From cp to Pa s
viscg = viscg/1000;
Area = pi * (d/2)^2;  % Cross section area [m]
usg = qg/Area;
usl = ql/Area;
um = usg + usl;
% Hs = usl/um;

% Liquid single phase:
DPDX = DPDX_liquid(d, usl, usg, deno, visco, angle);
if DPDX>0
DENS = deno;
HOLDUP = 1;
return
end

% Gas single phase:
DPDX = DPDX_gas(d, usl, usg, deng, viscg, angle);
if DPDX>0
DENS = deng;
HOLDUP = 0;
return
end

% Bubble flow:
Uoo = bubble_Uo(deno, deng, intpt, angle);
%alfa = bubble_H(usg, usl, Uoo);
% ULdb = usl / Hs;
% ULdf = usl / Hbb;
[DPDX, DENS, HOLDUP] = dpdx_bubbleflow(d, usl, usg, um, deno, deng, visco,
viscg, intpt, angle, Uoo);
if DPDX>0
    return;
end

% DENS_Mat is mixture density (with slip), HOLDUP_Mat is holdup with slip.

% Stratified flow:
HLD = stratified_HLD(d, Area, usl, usg, deno, deng, visco, viscg, angle);
% Is this with slip or without?
if HLD ~=5 % An error has occurred.
Hst = stratified_H(Area, d, HLD);
% ULst = usl / Hst;
UGst = usg / (1 - Hst);
DPDX = dpdx_stratified(d, Area, usl, deno, deng, viscg, angle, HLD, UGst);
end
if DPDX>0
DENS = HLD * deno + (1-HLD)*deng;
HOLDUP = HLD;
```

```
return;
end

% Annular flow:
TWODLD = annular_X_FP(d, Area, usl, usg, deno, deng, visco, viscg, angle);
if TWODLD ~=5 % An error has occurred.
DLDL = 0.5 * TWODLD;
Hannn = annular_H(Area, d, DLDL);
ULannn = usl / Hannn;
UGannn = usg / (1 - Hannn);
DPDX = dpdx_annular(DLDL, d, Area, usl, usg, deno, deng, visco, angle,
Hannn, ULannn, UGannn);
end
if DPDX>0
DENS = Hannn*deno + (1-Hannn)*deng;
HOLDUP = Hannn;
return;
end

% Slug flow:
Utt = slug_UT(d, usl + usg, angle);
Ubs = slug_Ub(usl + usg, deno, deng, intpt, angle);
Rsi = slug_Rsi(d, usl + usg, deno, visco, angle);
AlfaS = 1 - Rsi;
Ull = slug_UL(usl + usg, Ubs, AlfaS, Rsi);
Lss = 20 * d;
AlfaU = slug_AlfaU(usl, Ull, Rsi, Utt);
HfD = slug_stratified_HLD(d, Area, usl + usg, deno, deng, visco, viscg,
angle, Utt, Rsi, Ull);
TWODLSsl = slug_annular_X_FP(d, Area, usl, usg, deno, deng, visco, angle,
Utt, Ubs, AlfaS);
if HfD~=5 || TWODLSsl~=5 % Error message

if TWODLSsl == 0
    DLDsl = 0;
else
    DLDsl = TWODLSsl * 0.5;
end
Rf = slug_Rf(d, Area, angle, DLDsl, HfD);
Uff = slug_UF(Utt, Rf, Ull, Rsi);
Lu = slug_Lu(Lss, Ull, Rsi, Uff, Rf, usl);
[DPDX, HOLDUP] = dpdx_slug_unit(DLDsl, HfD, Area, d, deno, deng, angle,
AlfaS, AlfaU, Lss, Lu, usl + usg, visco, viscg, Uff);
DENS = HOLDUP*deno + (1-HOLDUP)*deng;
else
    disp('Flowrange out of bounds')
    return;
end

end
```

“frictionFactor.m”

```
function FZ = frictionFactor(d, e, Re)

FZ = 16/Re;
for i = 1:100
    fFZ = 3.48-4*log(2*e/d + 9.35/(Re*sqrt(FZ)))-1/sqrt(FZ);
    dfFZ = - (0.5*e*Re*sqrt(FZ)-4.06065*d*sqrt(FZ) +
2.3375*d)/(e*Re*FZ^2+4.675*d*FZ^(3/2));
    if abs(fFZ)<0.001
        break;
    end
    FZ = FZ - fFZ / dfFZ;    % Newton-Rhapson formula
    if FZ<0
        FZ = -FZ;
    end
end

end
```

“stratified_Xr.m”

```
function res = stratified_Xr(Xr, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG,
beta)
% Calculation of force balance function value for HLD=Xr
% See function stratified0 for definition of variables

    HLD = Xr;
    W = 2 * HLD - 1;
    acosW = acos(W);
    AL = 0.25 * d ^ 2 * (3.14 - acosW + W * (1 - W ^ 2) ^ 0.5);
    Ag = A - AL;
    SL = d * (3.14 - acosW);
    Sg = 3.14 * d - SL;
    SI = d * (1 - W ^ 2) ^ 0.5;
    DL = 4 * AL / SL;
    Dg = 4 * Ag / (Sg + SI);
    UL = ULS * A / AL;
    Ug = UGS * A / Ag;
   % alfa = Ag / (AL + Ag);
    REL = UL * DL * RHOL / VISL;
    Reg = Ug * Dg * RHOg / VISG;
    if REL < 1500
        Cl = 16;
        nL = 1 ;
    else
        Cl = 0.046;
        nL = 0.2;
    end
    if Reg < 1500
        CG = 16;
        ng = 1 ;
    else
        CG = 0.046;
        ng = 0.2;
    end
    TAUL = 0.5 * RHOL * UL ^ 2 * Cl * REL ^ (-nL);
```

```
    TAUg = 0.5 * RHOg * Ug ^ 2 * CG * Reg ^ (-ng);
    TAUi = 0.5 * RHOg * abs(Ug - UL) * (Ug - UL) * CG * Reg ^ (-ng);

    res = TAUg * Sg / Ag - TAUL * SL / AL + TAUi * SI * (1 / AL + 1 / Ag) -
(RHOL - RHOg) * 9.81 * sin(beta);
end
```

“stratified_HLD.m”

```
%Stratified_flow_liquidlevel

function res = stratified_HLD(d, A, ULS, UGS, RHOL, RHOg, VISL, VISG, beta)
% Calculation of liquid level for stratified flow
% The liquid level is calculated by finding the root of the force balance
function
% The root is located using a modified False Position method

% Checking if flow is single phase


    X0 = 0.01;
    X1 = 0.999;
    G0 = stratified0(X0, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG, beta);
    G1 = stratified1(X1, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG, beta);
    i = 0;
    I0 = 0;
    I1 = 0;
    Gr = 1;
    while i < 100 && abs(Gr) > 0.01
        Xr = X1 - G1 * (X0 - X1) / (G0 - G1);
        Gr = stratified_Xr(Xr, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG,
beta);
        Test = G0 * Gr;
        if (Test <= 0)
            X1 = Xr;
            G1 = Gr;
            I0 = I0 + 1;
            if (I0 > 2)
                G0 = G0 / 2;
            end
        else
            X0 = Xr;
            G0 = Gr;
            I1 = I1 + 1;
            if (I1 > 2)
                G1 = G1 / 2;
            end
        end
        i = i + 1;
    end

    if i>=100
         disp('Stratified switching to NR');
            X = 0.001;
    Gr = stratified0(X, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG, beta);

    i = 1;
```

```
    while abs(Gr) > 0.1 % Newton Raphson

      if i==1 % First iteration
          X_old = X;
          Gr_old = Gr;
          X = 0.999;
          Gr = stratified0(X, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG,
beta);
%           if Gr * Gr_old >0 % Same signe implies that no solution exists
%               disp('No solution for Stratified flow');
%               res = 5; % Error output
%               return;
%           end
      else
          temp = X;
          X = X - Gr/(Gr-Gr_old)*(X-X_old);
          X_old = temp;
          if X<0  % X is out of bounds
              X = X_old/2;
          elseif X>1
              X = (1+X_old)/2;
          end

          Gr_old = Gr;
          Gr = stratified0(X, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG,
beta);
      end
      if i>100
          disp('Stratified HLD failed to converge');
          res = 5; % Error output
          return;
      end
      i = i+1;
    end
    res = X;
    else
    res = Xr;
    end

end
```

“stratified_H.m”

```
% '######################################
% 'Stratified_liquid_holdup
% '######################################

function res = stratified_H(A, d, HLD)
% ' Calculation of liquid holdup for stratified flow:
% '
% '   HLD = liquid level derived from force balance
    W = 2 * HLD - 1;
    acosW = acos(W);
    AL = 0.25 * d ^ 2 * (3.14 - acosW + W * (1 - W ^ 2) ^ 0.5);
% '
% '   H = Liquid holdup
    H = AL / A;
    res = H;
end
```

“stratified1.m”

```
function res = stratified1(X1, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG,
beta)
% Calculation of force balance function value for HLD=X1
% See function stratified0 for definition of variables

    HLD = X1;
    W = 2 * HLD - 1;
    acosW = acos(W);
    AL = 0.25 * d ^ 2 * (3.14 - acosW + W * (1 - W ^ 2) ^ 0.5);
    Ag = A - AL;
    SL = d * (3.14 - acosW);
    Sg = 3.14 * d - SL;
    SI = d * (1 - W ^ 2) ^ 0.5;
    DL = 4 * AL / SL;
    Dg = 4 * Ag / (Sg + SI);
    UL = ULS * A / AL;
    Ug = UGS * A / Ag;
   % alfa = Ag / (AL + Ag);
    REL = UL * DL * RHOL / VISL;
    Reg = Ug * Dg * RHOg / VISG;
    if REL < 1500
        Cl = 16;
        nL = 1 ;
    else
        Cl = 0.046;
        nL = 0.2;
    end
    if Reg < 1500
        CG = 16;
        ng = 1 ;
    else
        CG = 0.046;
        ng = 0.2;
    end
    TAUL = 0.5 * RHOL * UL ^ 2 * Cl * REL ^ (-nL);
    TAUg = 0.5 * RHOg * Ug ^ 2 * CG * Reg ^ (-ng);
    TAUi = 0.5 * RHOg * abs(Ug - UL) * (Ug - UL) * CG * Reg ^ (-ng);

    res = TAUg * Sg / Ag - TAUL * SL / AL + TAUi * SI * (1 / AL + 1 / Ag) -
(RHOL - RHOg) * 9.81 * sin(beta);
end
```

“slug_UT.m”

```
% '######################################
% 'Slug_translational_velocity
% '######################################
function res = slug_UT(d, US, beta)
% 'Calculation of slug translation velocity:
% '
% 'Drift velocity:
    Ud = 0.35 * sqrt(9.81 * d) * sin(beta) + 0.54 * sqrt(9.81 * d) *
cos(beta);
% 'Translational velocity:
    cp = 1.2;
```

```
    UT = cp * US + Ud;
    res = UT;
end
```

“slug_UL.m”

```
% '#####################################
% 'Slug_liquid_velocity_slugzone
% '#####################################
function res = slug_UL(US, Ub, AlfaS, Rsi)
% 'Calculation of liquid velocity in liquid slug:
% '
    UL = (US - Ub * AlfaS) / Rsi;
    res = UL;
end
```

“slug_UF.m”

```
% '#####################################
% 'Slug_liquid_velocity_filmzone
% '#####################################

function res = slug_UF(UT, Rf, UL, Rsi)
% ' Calculation of liquid velocity in slug film zone:
% '
% ' Checking if liquid holdup has been calculated:
    if (Rf == 0)
        UF = 0;
    else
        UF = (UT * Rf - (UT - UL) * Rsi) / Rf;
    end
 res = UF;
end
```

“slug_Ub.m”

```
% '#####################################
% 'Slug_gas_velocity_slugzone
% '#####################################
function res = slug_Ub(US, RHOL, RHOg, SIG, beta)
% 'Calculation of gas velocity in liquid slug:
% '
% 'Bubble free rise velocity:
    Uo = 1.54 * (SIG * 9.81 * abs(RHOL - RHOg) / RHOL ^ 2) ^ 0.25;
% 'GAS VELOCITY in liquid slug:
    Ub = US + Uo * sin(beta);
    res = Ub;
end
```

“slug_stratified_HLD.m”

```
% '#####################################
% 'Slug_liquidlevel_stratified
% '#####################################
function res = slug_stratified_HLD(d, A, US, RHOL, RHOg, VISL, VISG, beta,
UT, Rsi, UL)
% 'Calculation of liquid level in slug film zone, modelled as stratified
flow
% 'The liquid level is calculated by finding the root of the force balance
function
% 'The root is located using a modified False Position method
% '
% 'Checking if flow is single phase

% 'Checking for slug flow : If no other flow regimes has been identified,
slug flow is assumed
% '    If (dpdx_bubble = "NA") And (dpdx_strat = "NA") And (dpdx_ann =
"NA") Then
% '
% ' Calculating slug parameters which are independent of liquid level, HLD:
% '
% '   Slug Reynolds number:
%       Res = US * RHOL * d / VISL;
% % '
% % '   CONSTANTS FOR CALCULATING FRICTION FACTOR:
%       CS = 0.046;
%       n = 0.2;
%
%         Fs = CS * (Res) ^ (-n);
% '
% ' Starting iteration on liquid level:
        X0 = 0.01;
        X1 = 0.99;
        G0 = slug_stratified0(X0, d, A, RHOL, RHOg, VISL, VISG, beta, US,
UT, UL, Rsi);
        G1 = slug_stratified0(X1, d, A, RHOL, RHOg, VISL, VISG, beta, US,
UT, UL, Rsi);
        Test = G0 * G1;
        if (Test > 0)
            X1 = 0.7;
         % Skjønner ikke hvorfor denne trengs   G1 = slug_stratified0(X1,
d, A, ULS, UGS, RHOL, RHOg, VISL, VISG, beta, US, UT, UL, Rsi, AlfaS); %
G1 = slug_stratified1(X1, D, A, ULS, UGS, RHOL, RHOg, VISL, VISg, BETA#,
US, UT, UL, Rsi, AlfaS)
        end
        i = 0;
        Gr = 1;
        I1 = 0;
        I0 = 0;
        while i < 100 && abs(Gr) > 0.01
            Xr = X1 - G1 * (X0 - X1) / (G0 - G1);
            Gr = slug_stratified0(Xr, d, A, RHOL, RHOg, VISL, VISG, beta,
US, UT, UL, Rsi);
            Test = G0 * Gr;
            if (Test < 0)
                X1 = Xr;
                G1 = Gr;
                I0 = I0 + 1;
                if (I0 > 2)
```

```
                    G0 = G0 / 2;
                end
            elseif (Test > 0)
                X0 = Xr;
                G0 = Gr;
                I1 = I1 + 1;
                if (I1 > 2)
                    G1 = G1 / 2;
                end
            end
            i = i + 1;
        end

        if i>=100
            disp('Slug stratified switching to NR');
                X = 0.001;
    Gr = slug_stratified0(X, d, A, RHOL, RHOg, VISL, VISG, beta, US, UT,
UL, Rsi);

    i = 1;
    while abs(Gr) > 0.1 % Newton Raphson

      if i==1 % First iteration
          X_old = X;
          Gr_old = Gr;
          X = 0.999;
          Gr = slug_stratified0(X, d, A, RHOL, RHOg, VISL, VISG, beta, US,
UT, UL, Rsi);
%           if Gr * Gr_old >0 % Same signe implies that no solution exists
%               disp('No solution for Slug Stratified flow');
%               res = 5; % Error output
%               return;
%           end
      else
          temp = X;
          X = X - Gr/(Gr-Gr_old)*(X-X_old);
          X_old = temp;
          if X<0  % X is out of bounds
              X = X_old/2;
          elseif X>1
              X = (1+X_old)/2;
          end
           Gr_old = Gr;
          Gr = slug_stratified0(X, d, A, RHOL, RHOg, VISL, VISG, beta, US,
UT, UL, Rsi);
      end
      if i>100
          disp('Slug Stratified HLD failed to converge');
          res = 5; % Error output
          return;
      end
      i = i+1;
    end
    res = X;
        else
        res = Xr;
        end
% '    Else
% '    slug_stratified_HLD = "NA"
% '    End If
 end
```

“slug_stratified0.m”

```
function res = slug_stratified0(X0, d, A, RHOL, RHOg, VISL, VISG, beta, US,
UT, UL, Rsi)
% 'Calculation of force balance function value for HLD=X0
% '
% ' HLD = liquid level (m)
% ' ALf = cross sectional area, liquid (m2)
    HLD = X0;
    WF = 2 * HLD - 1;
    acosWF = acos(WF);
    ALf = 0.25 * d ^ 2 * (3.14 - acosWF + WF * sqrt(1 - WF ^ 2));
% ' Rf  = Liquid holdup in film zone
% ' Agf = cross sectional area, gas (m2)
% ' SLF = interface circumference between liquid and pipe (m)
% ' Sgf = interface circumference between gas and pipe (m)
% ' Si  = interface circumference between gas and liquid
    Rf = ALf / A;
    Agf = A - ALf;
    SLF = d * (pi - acosWF);
    Sgf = 3.14 * d - SLF;
    SI = d * sqrt(1 - WF ^ 2);
% ' LIQUID VELOCITY IN FILM:
    UF = (UT * Rf - (UT - UL) * Rsi) / Rf;
% ' GAS VELOCITY IN FILM ZONE:
    Ug = (US - UF * Rf) / (1 - Rf);
% '
% ' DLF = hydraulic diameter, liquid film
% ' Dgf = hydraulic diameter, gas
    DLF = 4 * ALf / SLF;
    Dgf = 4 * Agf / (Sgf + SI);
% ' Reynolds no of liquid in liquid film:
    RelF = abs(UF) * DLF * RHOL / VISL;
% ' Reynolds no of gas in liquid film:
    Regf = abs(Ug) * Dgf * RHOg / VISG;
% '
% ' Blasius friction factor exponent and coefficient:
    if RelF < 1500
        Cl = 16;
        nL = 1 ;
    else
        Cl = 0.046;
        nL = 0.2;
    end
    if Regf < 1500
        CG = 16;
        ng = 1 ;
    else CG = 0.046;
        ng = 0.2;
    end
% '
% ' SHEAR STRESS BETWEEN LIQUID AND PIPE:
    TAULf = 0.5 * RHOL * abs(UF) * UF * Cl * RelF ^ (-nL);
% ' SHEAR STRESS BETWEEN GAS AND PIPE:
    TAUgf = 0.5 * RHOg * abs(Ug) * Ug * CG * Regf ^ (-ng);
% ' SHEAR STRESS BETWEEN LIQUID AND GAS:
    TAUIf = 0.5 * RHOg * abs(Ug - UF) * (Ug - UF) * CG * Regf ^ (-ng);
     res = TAUgf * Sgf / Agf - TAULf * SLF / ALf + TAUIf * SI * (1 / ALf +
1 / Agf) - (RHOL - RHOg) * 9.81 * sin(beta);
end
```

“slug_Rsi.m”

```
% '#####################################
% 'Slug_liquid_holdup_slugzone
% '#####################################
function res = slug_Rsi(d, US, RHOL, VISL, beta)
% 'Calculation of liquid holdup in liquid slug
% '
% 'Slug Reynolds number:
    Res = US * RHOL * d / VISL;
% 'Liquid holdup in slug:
    Rsi = 1 * exp(-(0.45 * beta + 2.48 * 10 ^ (-6) * Res));
    if (Rsi < 0.2)
        Rsi = 0.2;
    end
    res = Rsi;
end
```

“slug_Rf.m”

```
% '#####################################
% 'Slug_Liquid_holdup_filmzone
% '#####################################
function res = slug_Rf(d, A, beta, DLD, HLD)
% ' Calculation of liquid hold up in slug film zone:
% '
% '   Checking if film zone is to modelled as annular or stratified
        if (beta > 0.5) || (beta < (-1.5))
% '           Checking if film thickness has been calculated:
            if (DLD == 0)
                Rf = 0;
            else
% '               Liquid holdup in film zone, modelled as annular flow :
                Rf = 1 - (1 - 2 * DLD) ^ 2;
            end

        else
% '           Checking if liquid level has been calculated:
            if (HLD == 0)
                Rf = 0;
            else
                WF = 2 * HLD - 1;
                acosWF = acos(WF);
                ALf = 0.25 * d ^ 2 * (3.14 - acosWF + WF * sqrt(1 - WF ^
2));
% '               Liquid holdup in film zone, modelled as stratified flow:
                Rf = ALf / A;
            end
        end
    res = Rf;
end
```

“slug_Lu.m”

```
% '#####################################
% 'Slug_unit_length
% '#####################################

function res = slug_Lu(LS, UL, Rsi, UF, Rf, ULS)
% ' Calculation of slug unit length
% '
% ' Testing if liquid holdup has been calculated:
    if (Rf == 0)
        Lu = 0;
    else
% '   Calculate slug unit length:
        Lu = LS * (UL * Rsi - UF * Rf) / (ULS - UF * Rf);
    end
    if (Lu < LS) && (Lu > 0)
        Lu = LS;
    end
res = Lu;
end
```

“slug_annular_X_FP.m”

```
% '#####################################
% 'Slug_filmthickness_annular
% '#####################################
function res = slug_annular_X_FP(d, A, ULS, UGS, RHOL, RHOg, VISL, beta,
UT, Ub, AlfaS)
% 'Calculation of film thickness in slug film zone, modelled as annular
flow
% 'The film thickness is calculated by finding the root of the force
balance function
% 'The root is located using a modified False Position method
% '
% 'Checking if flow is single phase

% 'Checking for slug flow : If no other flow regimes has been identified,
slug flow is assumed


        X0 = 0.001;
        X1 = 0.99;
        G0 = slug_annular1(X0, d, A, ULS, UGS, RHOL, RHOg, VISL, beta, UT,
Ub, AlfaS);
        G1 = slug_annular1(X1, d, A, ULS, UGS, RHOL, RHOg, VISL, beta, UT,
Ub, AlfaS);
        Test = G0 * G1;
        if (Test > 0)
            X1 = 0.7;
            G1 = slug_annular1(X1, d, A, ULS, UGS, RHOL, RHOg, VISL, beta,
UT, Ub, AlfaS);
        end
        I0 = 0;
        I1 = 0;
        i = 0;
```

```
        Gr = 1;
        while i < 100 && abs(Gr) > 0.01
            Xr = X1 - G1 * (X0 - X1) / (G0 - G1);
            Gr = slug_annular1(Xr, d, A, ULS, UGS, RHOL, RHOg, VISL, beta,
UT, Ub, AlfaS);
            Test = G0 * Gr;
            if (Test < 0)
                X1 = Xr;
                G1 = Gr;
                I0 = I0 + 1;
                if (I0 > 2)
                    G0 = G0 / 2;
                end
            elseif (Test > 0)
                X0 = Xr;
                G0 = Gr;
                I1 = I1 + 1;
                if (I1 > 2)
                    G1 = G1 / 2;
                end
            end
            i = i + 1;
        end

        if i>=100
            disp('Slug annular switching to NR');
                X = 0.001;
    Gr = slug_annular1(X, d, A, ULS, UGS, RHOL, RHOg, VISL, beta, UT, Ub,
AlfaS);

    i = 1;
    while abs(Gr) > 0.1 % Newton Raphson

      if i==1 % First iteration
          X_old = X;
          Gr_old = Gr;
          X = 0.999;
          Gr = slug_annular1(X, d, A, ULS, UGS, RHOL, RHOg, VISL, beta, UT,
Ub, AlfaS);
%           if Gr * Gr_old >0 % Same signe implies that no solution exists
%               disp('No solution for Slug Annular flow');
%               res = 5; % Error output
%               return;
%           end
      else
          temp = X;
          X = X - Gr/(Gr-Gr_old)*(X-X_old);
          X_old = temp;
          if X<0  % X is out of bounds
              X = X_old/2;
          elseif X>1
              X = (1+X_old)/2;
          end

          Gr_old = Gr;
          Gr = slug_annular1(X, d, A, ULS, UGS, RHOL, RHOg, VISL, beta, UT,
Ub, AlfaS);
      end
      if i>100
          disp('Slug Annular HLD failed to converge');
          res = 5; % Error output
```

```
            return;
        end
        i = i+1;
    end
    res = X;

        else
        res = Xr;
        end


end
```

“slug_annular1.m”

```
function res = slug_annular1(X1, d, A, ULS, UGS, RHOL, RHOg, VISL, beta,
UT, Ub, AlfaS)
% 'Calculating the force balance function value for DLD=0.5*X1
% 'See function slug_annular0 for definition of variables
% '
    DLD = 0.5 * X1;
    Delta = DLD * d;
    Dgf = d - 2 * Delta;
    Agf = 3.14 / 4 * Dgf ^ 2;
    ALf = A - Agf;
    SLF = 3.14 * d;
    SI = 3.14 * Dgf;
    Rf = ALf / A;
    Rsi = 1 - AlfaS;
    UL = (ULS + UGS - Ub * AlfaS) / Rsi;
    UF = (UT * Rf - (UT - UL) * Rsi) / Rf;
    Ug = (ULS + UGS - UF * Rf) / (1 - Rf);
    DLF = 4 * ALf / SLF;
    RelF = abs(UF) * DLF * RHOL / VISL;
% '   Blasius friction factor coefficients and exponent:
       if RelF < 1500
           CT = 16;
           n = 1 ;
       else
           CT = 0.046;
           n = 0.2;
       end
    TAUL = CT * RelF ^ (-n) * 0.5 * RHOL * abs(UF) * UF;
    fi = 0.005 * (1 + 300 * Delta / d);
    TAUi = fi * 0.5 * RHOg * abs(Ug - UF) * (Ug - UF);
    g = -TAUL * SLF / ALf + TAUi * (SI / ALf + SI / Agf) - (RHOL - RHOg) *
9.81 * sin(beta);
    res = g;
end
```

“slug_AlfaU.m”

```
% '#####################################
% 'Slug_average_void_fraction
% '#####################################
function res = slug_AlfaU(ULS, UL, Rsi, UT)
% ' Calculation of average void fraction for slug unit:
% '
    AlfaU = (-ULS + UL * Rsi + UT * (1 - Rsi)) / UT;
    res = AlfaU;
end
```

“dpdx_stratified.m”

```
% '#####################################
% 'Stratified_flow_dPdx
% '#####################################

function DPDX = dpdx_stratified(d, A, ULS, RHOL, RHOg, VISG, beta, HLD, Ug)
% ' Testing for stratified flow regime and calculating pressure gradient
% '
% 'Checking if flow is bubble flow or single phase

% '   Calculations for stratified flow
    W = 2 * HLD - 1;
    acosW = acos(W);
    AL = 0.25 * d ^ 2 * (3.14 - acosW + W * (1 - (W) ^ 2) ^ 0.5);
    Ag = A - AL;
    SL = d * (3.14 - acosW);
    Sg = 3.14 * d - SL;
    SI = d * (1 - W ^ 2) ^ 0.5;
    UL = ULS * A / AL;
% ' Calculation of Kelvin Helmholtz instability criterion.
% ' Flow is stabilised when gravity dominates
    UgU = (1 - HLD) * (abs(RHOL - RHOg) * 9.81 * cos(beta) * Ag / (RHOg *
SI)) ^ 0.5;
% '
% ' Checking for stratified flow
    if (UgU > Ug) && (beta > -1.5) && (beta < 0.5)
% '    Calculate pressure gradient for stratified flow:
        Dg = 4 * Ag / (Sg + SI);
        Reg = Ug * Dg * RHOg / VISG;
        if Reg < 1500
            CG = 16;
            ng = 1 ;
        else
            CG = 0.046;
            ng = 0.2;
        end
        TAUg = 0.5 * RHOg * Ug ^ 2 * CG * Reg ^ (-ng);
        TAUi = 0.5 * RHOg * abs(Ug - UL) * (Ug - UL) * CG * Reg ^ (-ng);
       DPDX = TAUg * Sg / Ag + TAUi * SI / Ag + RHOg * 9.81 * sin(beta);
    else
        DPDX = 0;
    end

end
```

“dpdx_slug_unit.m”

```
% '#####################################
% 'Slug_flow_dpdx_global
% '#####################################
function [DPDX, holdup] = dpdx_slug_unit(DLD, HfD, A, d, RHOL, RHOg, beta,
AlfaS, AlfaU, LS, Lu, US, VISL, VISG, UF)
% 'Module for calculating pressure gradient for slug flow (intermittent
flow regime)
% '
% 'Checking for slug flow : If no other flow regimes has been identified,
slug flow is assumed

% '
% '   Calculate pressure drop using global momentum balance on slug unit:
% '   Parameters required for calculating TAUs:
% '
% '       RHOms = mixture density in slug
% '       VISms = mixture viscosity in slug
            RHOms = AlfaS * RHOg + (1 - AlfaS) * RHOL;
            VISms = AlfaS * VISG + (1 - AlfaS) * VISL;
% '
% '       Res = mixture Reynolds number in slug
            Res = US * RHOms * d / VISms;

% '       Blasius friction factor coefficients and exponent:
            if Res < 1500
                CS = 16;
                ns = 1;
            else
                CS = 0.046;
                ns = 0.2;
            end

% '       fs = friction factor, slug
            Fs = CS * Res ^ (-ns);

% '       TAUs = shear stress in slug zone
            TAUs = Fs * 0.5 * RHOms * US ^ 2;

% '   Parameters required for calculating TAUF and TAUG:
%
% '       Checking if film zone is to modelled as annular or stratified
        if (beta > 0.5) || (beta < (-1.5))
% '           Film zone modelled as annular flow:
            Rf = 1 - (1 - 2 * DLD) ^ 2;
            Delta = DLD * d;
            Dgf = d - 2 * Delta;
            Agf = 3.14 / 4 * Dgf ^ 2;
            ALf = A - Agf;
            SLF = 3.14 * d;
            Sgf = 0;
        else
% '           Film zone modelled as stratified flow:
            WF = 2 * HfD - 1;
            acosWF = acos(WF);
            ALf = 0.25 * d ^ 2 * (3.14 - acosWF + WF * sqrt(1 - WF ^ 2));
            Rf = ALf / A;
            Agf = A - ALf;
```

```
            SLF = d * (3.14 - acosWF);
            Sgf = 3.14 * d - SLF;
            SI = d * sqrt(1 - WF ^ 2);
            Dgf = 4 * Agf / (Sgf + SI);
        end

            DLF = 4 * ALf / SLF;
            Ug = (US - UF * Rf) / (1 - Rf);

% '       Reynolds no of liquid in liquid film:
            RelF = abs(UF) * DLF * RHOL / VISL;
% '       Reynolds no of gas in liquid film zone:
            Regf = abs(Ug) * Dgf * RHOg / VISG;

            if RelF < 1500
                Cl = 16;
                nL = 1;
            else
                Cl = 0.046;
                nL = 0.2;
            end
            if Regf < 1500
                CG = 16;
                ng = 1;
            else
                CG = 0.046;
                ng = 0.2;
            end

% '       SHEAR STRESS BETWEEN LIQUID AND PIPE:
            TAULf = 0.5 * RHOL * abs(UF) * UF * Cl * RelF ^ (-nL);
% '       SHEAR STRESS BETWEEN GAS AND PIPE:
            if (Sgf == 0)
                TAUgf = 0;
            else
                TAUgf = 0.5 * RHOg * abs(Ug) * Ug * CG * Regf ^ (-ng);
            end

% '       Mixture density in slug unit:
            RHOmu = (1 - AlfaU) * RHOL + AlfaU * RHOg;

      DPDX = RHOmu * 9.81 * sin(beta) + TAUs * (3.14 * d / A) * LS / Lu +
(TAULf * SLF + TAUgf * Sgf) / A * (Lu - LS) / Lu;
      holdup = 1-AlfaU;
end
```

“DPDX_liquid.m”

```
function DPDX = DPDX_liquid(d, ULS, UGS, RHOL, VISL, beta)

% Single Phase liquid
global rough
%Testing for single phase liquid flow, and calculation of pressure gradient

if (ULS > 0)

    if UGS>0 % Gas flowing
        DPDX = 0;
        return
    end

% REl = Reynolds number for liquid (oil)
    REL = RHOL * ULS * d / VISL;
    if (REL > 2000)
        FZ = frictionFactor(d, rough, REL);
        else
        FZ = 16 / REL;
    end
% Checking for single phase liquid flow
%    Calculate pressure gradient for single phase liquid (Pa/m):
        DPDX = RHOL * 9.81 * sin(beta) + 2 * FZ * RHOL * ULS ^ 2 / d;


else
    DPDX = 0;
end


end
```

“DPDX_gas.m”

```
function DPDX = DPDX_gas(d, ULS, UGS, RHOg, VISG, beta)

% Single Phase liquid
global rough
%Testing for single phase liquid flow, and calculation of pressure gradient

if (UGS > 0)

    if ULS>0 % Liquid flowing
        DPDX = 0;
        return
    end

% REl = Reynolds number for liquid (oil)
    REL = RHOg * UGS * d / VISG;
    if (REL > 2000)
        FZ = frictionFactor(d, rough, REL);
        else
        FZ = 16 / REL;
```

```
    end
% Checking for single phase liquid flow
%    Calculate pressure gradient for single phase liquid (Pa/m):
        DPDX = RHOg * 9.81 * sin(beta) + 1/2 * FZ * RHOg * UGS ^ 2 / d;

else
    DPDX = 0;
end


end
```

“dpdx_bubbleflow.m”

```
%Bubble_flow

function [DPDX, rhom, holdup] = dpdx_bubbleflow(d, ULS, UGS, US, RHOL,
RHOg, VISL, VISG, SIG, beta, Uo)
%Module for testing on bubble flow regime and calculating pressure gradient
global rough


%Calculations for dispersed bubble flow:

%   Alfa = gas void fraction, no-slip
        alfa = UGS / US;
%   VISM = mixture viscosity (Pa-s)
%   RHOm = mixture density (kg/m3)
%   ED   = dimensionless pipe roughness
        VISM = VISL * (1 - alfa) + VISG * alfa;
        rhom = RHOL * (1 - alfa) + RHOg * alfa;

%   REmx = Reynolds number of mixture
        REmx = US * d * rhom / VISM;
%   Calculation of friction factor
        if (REmx > 2000)
           Frm = frictionFactor(d, rough, REmx);
        else
           Frm = 16 / REmx;
        end

%   DBMX = critical bubble diameter
        DBMX = (0.725 + 4.15 * (alfa) ^ 0.5) * (SIG / RHOL) ^ 0.6 * (2 *
Frm / d * US ^ 3) ^ (-0.4);
        COSB = cos(beta);
        if (COSB < 0.000001)
            COSB = 0.000001;
        end


%    DCD = critical bubble size below which bubble are prevented from
deformation
%    DCB = critical bubble size below which bubble migration to upper part
of pipe is prevented
        DCD = 2 * (0.4 * SIG / (abs(RHOL - RHOg) * 9.81)) ^ 0.5;
```

```
        DCB = 0.375 * (RHOL / (RHOL - RHOg)) * Frm * US ^ 2 / (9.81 *
COSB);
        if (DCB > DCD)
            DCB = DCD;
        end

% Calculations for bubbly flow:

%   Uo = bubble rise velocity
        if abs(Uo) < 0.00001
            Uo = 0.00001;
        end

        MM = Uo + UGS + ULS;
        Alfab = (MM - (MM ^ 2 - 4 * Uo * UGS) ^ 0.5) / (2 * Uo);
dispersed_bubble = false;
% Checking for dispersed bubble flow:
%   Dispersed bubble flow if critical bubble diameter < DCB and void
fraction <0.52
        if (DBMX < DCB) && (alfa < 0.52)
          dispersed_bubble = true;

%   Checking for bubbly flow:
        elseif (Alfab < 0.25) && (beta > 1.3)
          bubbly_flow = true;
          alfa = Alfab;
          rhom = RHOL * (1 - alfa) + RHOg * alfa;
          %   REmx = Reynolds number of mixture

          REmx = US * d * rhom / VISM;
%   Calculation of friction factor
            if (REmx > 2000)
                Frm = frictionFactor(d, rough, REmx);
            else
                Frm = 16 / REmx;
            end

        else
           DPDX = 0;
           holdup = 0;
           rhom = 0;
           return;
        end

        if (dispersed_bubble == true) || (bubbly_flow == true)

%   Calculation of pressure gradient for bubble flow (dispersed or bubbly):

%   Recalculating mixture density using final void fraction:

            holdup = 1-alfa;
% %   W  = mass flow rate
% %   G  = mass flux
% %   XG = mass fraction of gas (quality)
%             W = ULS * A * RHOL + UGS * A * RHOg;
%             g = W / A;
%             XG = UGS * A * RHOg / W;
            DPDX = 2 / d * Frm * rhom * US ^ 2 + rhom * 9.81 * sin(beta);
        end
```

```
%   Alternative dpdx-equation including an acceleration term:
%   NOTE: Pressure, P, must be included in the function definition when
using the equation below.
%        dpdx_bubbleflow = (2 / D * Frm * rhom * Us ^ 2 + rhom * 9.81 *
Sin(BETA#)) / (1 - G ^ 2 * XG / (RHOg * P))

end
```

“dpdx_annular.m”

```
% '#######################################
% 'Annular_flow_dpDx
% '#######################################
function DPDX = dpdx_annular(DLD, d, A, ULS, UGS, RHOL, RHOg, VISL, beta,
hL, UL, Ug)
% 'Module for testing on annular flow regime and calculating pressure
gradient
% '
% 'Checking if flow is stratified,bubble or single phase
if (ULS > 0) && (UGS > 0)

% 'Calculations for annular flow
% '
% '   DLD = film thickness/diameter
% '   DELTA = film thickness (m)
% '   DG  = hydraulic diameter, gas (m)
% '   DL  = hydraulic diameter, liquid (m)
% '   Ag  = cross sectional area, gas (m2)
% '   Al  = cross sectional area, liquid (m2)
% '   Sl  = liquid circumference (m)
% '
    Delta = DLD * d;
    Dg = d - 2 * Delta;
    Ag = 3.14 / 4 * Dg ^ 2;
    AL = A - Ag;
    SL = 3.14 * d;
    DL = 4 * AL / SL;
% '
% ' ReL = Reynolds number of liquid
    REL = UL * DL * RHOL / VISL;
% ' Blasius friction factor coefficient and exponent:
    if REL < 1500
        CT = 16;
        n = 1;
    else
        CT = 0.046;
        n = 0.2;
    end
% '
% ' Fi = friction factor, interface
    fi = 0.005 * (1 + 300 * Delta / d);
% ' TAUi = shear stress, interface
    TAUi = fi * 0.5 * RHOg * abs(Ug - UL) * (Ug - UL);
% '
% ' Criterion for liquid film stability:
```

```
    STAB = 9.81 * (RHOL - RHOg) * d * sin(beta) * ((1 - 2 * DLD) ^ 2 - 2 *
(DLD - DLD ^ 2)) - 1 / 16 * CT * RHOL * (d * RHOL / VISL) ^ (-n) * ULS ^ (2
- n) * ((DLD - DLD ^ 2) + (1 - 2 * DLD) ^ 2) / (DLD - DLD ^ 2) ^ 3;
% ' Criterion for spontaneous blockage: Liquid holdup < 0.24
% '
% ' Checking for annular flow:
    if (STAB < 0) && (hL < 0.24)
% '
% ' Calculate pressure gradient for annular flow:
        DPDX = (4 * TAUi / (d - 2 * Delta)) + RHOg * 9.81 * sin(beta);

    else
    DPDX = 0;
    end
else
    DPDX = 0;
end
end
```

“annular_X_FP.m”

```
% '#######################################
% 'Annular_flow_filmthickness
% '#######################################
function res = annular_X_FP(d, A, ULS, UGS, RHOL, RHOg, VISL, VISG, beta)
% ' Calculation of film thickness in annular flow using force balance
% ' The root of the force balance function is determined useing a modified
False Position method.
% '
% 'Checking if flow is single phase

    X0 = 0.01;
    X1 = 0.99;
    G0 = annular0(X0, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG, beta);
    G1 = annular0(X1, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG, beta); % Used
annular1 before. This is the exact same as annular0.
    i = 0;
    I0 = 0;
    I1 = 0; % Is this the correct use of the variable?
    Gr = 1;
    while i < 100 && abs(Gr) > 0.01
        Xr = X1 - G1 * (X0 - X1) / (G0 - G1);
        Gr = annular0(Xr, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG, beta);
        Test = G0 * Gr;
        if (Test < 0)
            X1 = Xr;
            G1 = Gr;
            I0 = I0 + 1;
            if (I0 > 2)
                G0 = G0 / 2;
            end
        elseif (Test > 0)
                X0 = Xr;
                G0 = Gr;
                I1 = I1 + 1;
                if (I1 > 2)
                    G1 = G1 / 2;
                end
```

```
        end
        i = i + 1;
    end
    if i>=100
         disp('Annular switching to NR');
           X = 0.001;
    Gr = stratified0(X, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG, beta);

    i = 1;
    while abs(Gr) > 0.1 % Newton Raphson

      if i==1 % First iteration
          X_old = X;
          Gr_old = Gr;
          X = 0.999;
          Gr = stratified0(X, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG,
beta);
%           if Gr * Gr_old >0 % Same signe implies that no solution exists
%               disp('No solution for Annular flow');
%               res = 5; % Error output
%               return;
%           end
      else
          temp = X;
          X = X - Gr/(Gr-Gr_old)*(X-X_old);
          X_old = temp;
          if X<0  % X is out of bounds
              X = X_old/2;
          elseif X>1
              X = (1+X_old)/2;
          end

          Gr_old = Gr;
          Gr = stratified0(X, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG,
beta);
      end
      if i>100
          disp('Annular flow failed to converge');
          res = 5; % Error output
          return;
      end
      i = i+1;
    end
    res = X;
    else
    res = Xr;
    end
end
```

“annular_H.m”

```
% '#####################################
% 'Annular_liquid_holdup
% '#####################################
function res = annular_H(A, d, DLD)
% 'Calculation of liquid holdup in annular flow, using film thickness
derived from force balance
% '
% ' DLD = film thickness/diameter
% ' DELTA = film thickness (m)
% ' DG  = hydraulic diameter, gas (m)
% ' Ag  = cross sectional area, gas (m2)
% ' Al  = cross sectional area, liquid (m2)
% ' Hl  = liquid holdup

    Delta = DLD * d;
    Dg = d - 2 * Delta;
    Ag = 3.14 / 4 * Dg ^ 2;
    AL = A - Ag;
    hL = AL / A;
    res = hL;
end
```

“annular_0.m”

```
function res = annular0(X0, d, A, ULS, UGS, RHOL, RHOg, VISL, VISG, beta)
% 'Calculating the force balance function value for DLD=0.5*X0
% '
% ' DLD = film thickness/diameter
% ' DELTA = film thickness (m)
% ' DG  = hydraulic diameter, gas (m)
% ' DL  = hydraulic diameter, liquid (m)
% ' Ag  = cross sectional area, gas (m2)
% ' Al  = cross sectional area, liquid (m2)
% ' Sl  = liquid circumference (m)
% ' Si  = gas/liquid interface circumference (m)
% ' UL  = liquid velocity (m/s)
% ' Ug  = gas velocity (m/s)
    DLD = 0.5 * X0;
    Delta = DLD * d;
    Dg = d - 2 * Delta;
    Ag = 3.14 / 4 * Dg ^ 2;
    AL = A - Ag;
    SL = 3.14 * d;
    SI = 3.14 * Dg;
    UL = ULS * A / AL;
    Ug = UGS * A / Ag;
    DL = 4 * AL / SL;
% '
    REL = UL * DL * RHOL / VISL;
% ' Blasius friction factor coefficients and exponent:
    if REL < 1500
        CT = 16;
        n = 1 ;
    else
        CT = 0.046;
```

```
        n = 0.2;
    end
% ' Liquid shear stress:
    TAUL = CT * REL ^ (-n) * 0.5 * RHOL * UL ^ 2;
% ' Interfacial friction factor:
    fi = 0.005 * (1 + 300 * Delta / d);
% ' Gas/liquid interface shear stress:
    TAUi = fi * 0.5 * RHOg * abs(Ug - UL) * (Ug - UL);
% ' Force balance function value:
    g = -TAUL * SL / AL + TAUi * (SI / AL + SI / Ag) - (RHOL - RHOg) * 9.81
* sin(beta);
    res = g;
end
```

“Mass_Balance.m”

```
function [pR_2, IOIP_2, IGIP_2, IWIP_2] = Mass_Balance(qo, qg, qw, pR, TR,
IOIP, IGIP, IWIP)

global GOR_0

Prod_Days = 350/2;
Np = qo*Prod_Days;
Gp = qg*Prod_Days;
Wp = qw*Prod_Days;
Cr = 10^-5*14.5;
Cw = 10^-4;

m = IGIP/IOIP; % At initial conditions
Sw = IWIP/(IWIP+IGIP+IOIP);  % At initial conditions

f_new = 1;
pR_new = pR-5;
while abs(f_new)>0.0000001

[Bo, Bw, Bg, Rs] = local_properties_MB(pR, TR, qo, qw, qg, 0);  % Witout
GL, At initial conditions
[Bo_2, Bw_2, Bg_2, Rs_2] = local_properties_MB(pR_new, TR, qo, qw, qg, 0);
% Witout GL, At new conditions

F = Np*(Bo_2-Rs_2*Bg_2)+Wp*Bw_2+Gp*Bg_2;
Eo = (Bo_2-Bo)+(Rs-Rs_2)*Bg_2;
Eg = Bo*(Bg_2/Bg-1);
Efw = -(1+m)*Bo*(Cr+Cw*Sw)*(pR_new-pR)/(1-Sw);
% Newton Raphson:
f_new = IOIP*(Eo+m*Eg+Efw)+IGIP*Bg_2-F;

if pR_new == pR-5
    f_old = f_new;
    pR_old = pR_new;
    pR_new = pR - 2.5;
else
    df = f_new - f_old;
    f_old = f_new;
    temp = pR_new;
    pR_new = pR_new - f_new/df*(pR_new - pR_old);
    pR_old = temp;
```

```
end
end
pR_2 = pR_new;

IOIP_2 = IOIP - Np;
if GOR_0<Rs
    Rs = GOR_0;
end
IGIP_2 = IGIP - (GOR_0-Rs)*Gp;
IWIP_2 = IWIP - Wp;

end
```

“local_properties_MB.m”

```
function [Bo, Bw, Bg, Rs] = local_properties_MB(p, T, qo_sc, qw_sc, qg_sc,
qgl_sc)

global gamma_g_0 gamma_gl_0 visc_g_sc_0 GOR_0 R dens_o_sc dens_w_sc p_sc
T_sc M_air bara

if qgl_sc~=0
    qg_tot_sc = qg_sc + qgl_sc;
    GOR = qg_tot_sc/qo_sc;
    gamma_g = (qg_sc*gamma_g_0 + qgl_sc*gamma_gl_0)/(qg_tot_sc);
    pBP = bubble_point(GOR, T, gamma_g);
    visc_g_sc = dead_visc_gas(gamma_g);
else  % No GL
    qg_tot_sc = qg_sc;
    GOR = GOR_0;
    gamma_g = gamma_g_0;
    pBP = bubble_point(GOR, T, gamma_g);
    visc_g_sc = visc_g_sc_0;
end
Rs = solution_GOR(p, T, gamma_g, pBP);
[z, Ppr, Tpr] = z_Standing(p, T, gamma_g);
dens_g_sc = gamma_g*M_air*p_sc*bara/(R*T_sc);
[Bo, Bw, Bg] = FVF (gamma_g, Rs, p, T, z, pBP);




end
```