```
classdef TubingObj < handle
%TubingObj. Tubing Object
% TubingObj represents a tubing inside a single well model. It allows
% calculation of pressure/temperature profile along the tubing.
%
%   Properties
%    - Qi              : flow rate tubing inlet           [m3/d]
%    - Qo              : flow rate tubing outlet          [m3/d]
%    - Pi              : pressure tubing outlet           [kPa]
%    - Po              : pressure tubing inlet            [kPa]
%    - Ti              : temperature tubing inlet         [C]
%    - To              : temperature tubing outlet        [C]
%    - TVD             : true vertical depth              [m]
%    - tTgprof         : tubing geothermal gradient       [K/m]
%    - tln             : tubing length                    [m]
%    - tdi             : tubing internal diameter         [m]
%    - trg             : tubing roughness                 [m]
%    - tuc             : tubing global heat coefficient   [kW/m2.K]
%    - tin             : tubing inclination               [rad]
%    - tsl             : tubing segment length            [m]
%    - BOmodel         : black oil model
%    - CalculationType : {ascending | descending}
%
%   Constant properties
%    - g               : gravity acceleration             [m/s2]
%
%   Dependent properties
%    - status          : programming property (to be used in further
 versions)
%    - Tprofile        : tubing temperature profile       [C]
%    - Pprofile        : tubing pressure profile          [kPa]
%    - tdepthp         : tubing depth profile             [m]
%
%   Methods
%    - SolveTubing
%        Depending on the calculation type, calculates P, T and Q on
 the
%        other tubing nozzle.
%    - PlotPprofile
%        Plotting pressure profile along the tubing depth.
%    - PlotTprofile
%        Plotting temperature profile along the tubing depth.
%    - PlotPTprofile
%        Plotting temperature and pressure profile along the ubing
 depth.
%
 ------------------------------------------------------------------------
% Development
%   By: Ruben Ensalzado
%       2015
%   Rev 00 151216 original release
%   Rev 01 160110 removing property: Zo, Zi
```

```
%   Rev 02 160531 including 'descending' calculations
%                 bug fixing: sl defintion @mpiping

    properties
        BOmodel
        Qi
        Qo
        Pi
        Po
        Ti
        To
        TVD
        tTgprof
        tln
        tdi
        trg
        tuc
        tin
        tsl
        CalculationType
    end

    properties (Constant = true)
        g = 9.81;
    end

    properties (Dependent = true)
        status
        Tprofile
        Pprofile
        tdepthp
    end

    methods (Hidden = true)
        % -- initialization method --

        function TU = TubingObj
            TU.CalculationType = 'ascending';
            TU.tsl = 2;
        end
    end

    methods
        % -- error verification methods --

        function set.CalculationType(TU, uType)
            if strcmpi(uType, 'ascending') ||
 strcmpi(uType, 'descending')
                TU.CalculationType = uType;
            else
                errrm = ['The available types are:';...
                         '- ascending             ';...
                         '- descending            '];
```

```
                error('TubingObj:BadArgument', ...
                      '%s \n\t%s \n\t%s \n', ...
                      errrm(1, :), errrm(2, :), errrm(3, :))
            end
        end

        function set.BOmodel(TU, uBO)
            if strcmpi(class(uBO), 'BOObj')
                TU.BOmodel = uBO;
            else
                error('TubingObj:BadArgument', ...
                      'The fluid model object has to be class BOObj')
            end
        end
    end

    methods
        % -- dependent properties --

        function status = get.status(TU)
            %Status. Independent property verification

            req = zeros(9, 1);

            req(1) = isempty(TU.Qi) && isempty(TU.Qo);
            req(2) = isempty(TU.Pi) && isempty(TU.Po);
            req(3) = isempty(TU.Ti) && isempty(TU.To);
            req(4) = isempty(TU.tln);
            req(5) = isempty(TU.tTgprof);
            req(6) = isempty(TU.tdi);
            req(7) = isempty(TU.trg);
            req(8) = isempty(TU.tuc);
            req(9) = isempty(TU.TVD);

            if all(~req)
                status = true;
            else
                status = false;
            end
        end

        function Tprofile = get.Tprofile(TU)
            %TubingTemperatureProfile
            %Units
            % T0 : C

            switch lower(TU.CalculationType)
                case 'ascending'
                    T0 = TU.Ti + 273.15;
                case 'descending'
                    T0 = TU.To + 273.15;
            end

             [piping, dy, n] = mpiping(TU);
```

```
            props = mprops(TU, n, 'Tprofile');
            [dtm, dti] = dtmatrix(TU, T0, n, dy, props, piping);
            Tprofile = dtm\dti;
        end

        function Pprofile = get.Pprofile(TU)
            %TubingPressureProfile.
            %Units
            % P0       : kPa
            % Pprofile : kPa

            switch lower(TU.CalculationType)
                case 'ascending'
                    P0 = TU.Pi;
                case 'descending'
                    P0 = TU.Po;
            end

            [piping, dy, n] = mpiping(TU);
            props = mprops(TU, n, 'Pprofile');
            [dpm, dpi] = dpmatrix(TU, P0, n, dy, props, piping);

            Pprofile = (dpm\dpi)/1e3;
        end

        function tdepthp = get.tdepthp(TU)
            %TubingDepthProfile.
            %Units
            % tln : m
            % tsl : m
            % TVD : m

            sn = round(TU.tln/TU.tsl) + 1;
            tdepthp = linspace(TU.TVD - TU.tln, TU.TVD, sn + 1)';
        end
    end

    methods
        function SolveTubing(TU)
            switch lower(TU.CalculationType)
                case 'ascending'
                    TU.Po = TU.Pprofile(end);
                    TU.To = TU.Tprofile(end) - 273.15;
                case 'descending'
                    TU.Pi = TU.Pprofile(1);
                    TU.Ti = TU.Tprofile(1) - 273.15;
            end
        end
    end

    methods (Hidden = true)
        % -- auxiliary functions --

        function ff = colebrook(~, Re, edr)
```

```
            %FictionFactor. Colebrook and White (1931)

            if Re <= 2000
                ff = 64/Re;
            else
                fff = @(f) (2/log(10))*log(edr/3.7 + 2.51/
(Re*sqrt(f))) + 1/sqrt(f);
                dff = @(f) -((2/log(10)*(2.51*0.5/Re)*(edr/3.7 + 2.51/
(Re*sqrt(f)))^(-1)*f^(-1.5)) + 0.5*f^(-1.5));

                eff = 1;
                ff0 = 1e-3;

                while eff > 1e-8
                    ff = ff0 - fff(ff0)/dff(ff0);
                    eff = abs(ff - ff0);
                    ff0 = ff;
                end
            end
        end

        function [dtm, dti] = dtmatrix(TU, T0, n, dy, props, piping)
            %DTmatrix. Temperature profile linear system matrix

            % rho : density
            % cpf : heat capacity
            % Tin : medium temperature

            % di : tubing internal diameter
            % uc : tubing U coefficient
            % qf : fluid volumetric flowrate

            rho = props(2:end, 1);      % kg/m3
            cpf = props(2:end, 2);      % kJ/kg.K
            Tin = props(2:end, 3);      % K

            di = piping(2:end, 1);      % m
            uc = piping(2:end, 3);      % kW/m2.K
            qf = piping(2:end, 4);      % m3/s
            mf = rho.*qf;               % kg/s
            ai = dy*pi*di;              % m2

            dti = zeros(n + 1, 1);

            switch lower(TU.CalculationType)
                case 'ascending'

                    dtm = sparse([2:n+1 2:n+1], [1:n 2:n+1], [(ai.*uc
 - 2*mf.*cpf) ...
                                (ai.*uc + 2*mf.*cpf)]);
                    dtm(1, 1) = 1;
                    dti(1) = T0;
                    dti(2:n+1) = 2*ai.*uc.*Tin;
```

```
                case 'descending'

                    dtm = sparse([1:n 1:n], [1:n 2:n+1], [(ai.*uc -
2*mf.*cpf) ...
                                (ai.*uc + 2*mf.*cpf)]);
                    dtm(end, end) = 1;
                    dti(end) = T0;
                    dti(1:end-1) = 2*ai.*uc.*Tin;
            end

        end

        function [dpm, dpi] = dpmatrix(TU, P0, n, dy, props, piping)
            %DPmatrix. Pressure profile linear system matrix

            % rho : density
            % vnu : dynamic viscosity

            % di : tubing internal diameter
            % rg : tubing roughness (E)
            % um : fluid velocity
            % in : tubing inclination
            % ga : gravity acceleration

            % ted : tubing relative roughness (E/di)
            % ff  : friction factor
            % tRe : Reynolds number

            rho = props(2:end, 1);       % kg/m3
            vnu = props(2:end, 2)./rho;  % m2/s

            di = piping(2:end, 1);       % m
            rg = piping(2:end, 2);       % m
            um = piping(2:end, 5);       % m/s
            in = cos(piping(2:end, 7));  % -
            ga = TU.g;                   % m/s2

            ted = rg./di;                % -
            ff = zeros(n , 1);           % -
            tRe = um.*di./vnu;           % -

            for i = 1:n
                ff(i) = colebrook(TU, tRe(i), ted(i));
            end

            dpi = zeros(n + 1, 1);

            switch lower(TU.CalculationType)
                case 'ascending'

                    dpm = sparse([2:n+1 1:n+1], [1:n 1:n+1], ...
                                 [-ones(1, n), ones(1, n + 1)]);
                    dpi(1) = P0*1000;     % Pa
```

```
                    dpi(2:end) = -dy*rho.*(ga.*in + ff.*um.^2./
(2*di));

                case 'descending'

                    dpm = sparse([1:n 1:n+1], [1:n 1:n+1], ...
                                 [-ones(1, n), ones(1, n + 1)]);
                    dpi(end) = P0*1000;   % Pa
                    dpi(1:end-1) = -dy*rho.*(ga.*in + ff.*um.^2./
(2*di));
            end
        end

        function [piping, dy, sn] = mpiping(TU)
            %Mpiping. Piping parameter values

            % ln : tubing length
            % di : tubing internal diameter
            % uc : tubing U coeffiecient
            % rg : tubing roughness
            % in : inclination
            % sl : minimum length of piping segment
            % sn : number of segments
            % dy : length of piping segment

            switch lower(TU.CalculationType)
                case 'ascending'
                    Q = TU.Qi;
                case 'descending'
                    Q = TU.Qo;
            end

            ln = TU.tln;                                % m
            di = TU.tdi;                                % m
            uc = TU.tuc;                                % kW/m2.K
            rg = TU.trg;                                % m
            in = TU.tin;                                % rad
            sl = TU.tsl;                                % m
            sn = round(ln/sl) + 1;
            dy = ln/sn;                                 % m

            piping = zeros(sn + 1, 6);

            piping(:, 1) = di;                          % m
            piping(:, 2) = rg;                          % m
            piping(:, 3) = uc;                          % kW/m2.K
            piping(:, 4) = Q/(3600*24);                 % m3/s
            piping(:, 5) = Q/(900*24*pi*di^2);          % m/s
            piping(:, 6) = TU.tdepthp;                  % m
            piping(:, 7) = in;                          % -
        end

        function props = mprops(TU, n, type)
            %Mprops. Fluid property values
```

```
    % emurho : density emulsion
    % emuCp  : specific heat capacity emulsion
    % emuvnu : dynamic viscosity emulsion

    switch lower(TU.CalculationType)
        case 'ascending'
            T = TU.Ti + 273.15;
        case 'descending'
            T = TU.To + 273.15;
    end

    switch type
        case 'Tprofile'
            gt = TU.tTgprof;
            depthp = TU.tdepthp;

            props = zeros(n + 1, 3);
            props(:, 1) = emurho(TU)*16.0186;    % kg/m3
            props(:, 2) = emuCp(TU)*4.2216;      % kJ/kg.K
            props(:, 3) = (depthp - depthp(1))*gt...
                          + T;                   % K
        case 'Pprofile'
            props = zeros(n + 1, 2);
            props(:, 1) = emurho(TU)*16.0186;    % kg/m3
            props(:, 2) = emuvnu(TU)/1e3;        % Pa.s
    end
end

function mCp = emuCp(TU)
    %SpecificHeat. (emulsion) Average with weight fraction
    %Units
    % mCp : BTU/lb.R

    switch lower(TU.CalculationType)
        case 'ascending'
            qo = TU.Qi;
        case 'descending'
            qo = TU.Qo;
    end

    Fo = qo*TU.BOmodel.orho;
    Fw = qo*(TU.BOmodel.WC/1e2)*TU.BOmodel.wrho;

    mCp = (TU.BOmodel.oCp*Fo + TU.BOmodel.wCp*Fw)/(Fw + Fo);
end

function mrho = emurho(TU)
    %Density. (emulsion) Mass balance
    %Units
    % mrho : lb/ft3

    switch lower(TU.CalculationType)
        case 'ascending'
```

```
                    qo = TU.Qi;
                case 'descending'
                    qo = TU.Qo;
            end

            qw = qo*TU.BOmodel.WC/1e2;
            mrho = (qo*TU.BOmodel.orho + qw*TU.BOmodel.wrho)/(qo +
 qw);
        end

        function mnu = emuvnu(TU)
            %Viscosity. (emulsion) Richardson model
            %Units
            % mnu : cP

            tTprofile = TU.Tprofile;
            n = size(tTprofile, 1);
            mnu = zeros(n, 1);

            for i = 1:n
                TU.BOmodel.T = 1.8*tTprofile(i) - 460;
                mnu(i) = TU.BOmodel.emu;
            end
        end
    end

    methods
       % -- information and plotting methods --

        function PlotPprofile(TU)
           %PlotPprofile. Plot pressure - depth
           % Plotting the pressure profile of the well along its depth

           if ~TU.status
               error('TubingObj:NotEnoughArguments', ...
                     'The tubing element must be properly defined')
           end

           figure
           axes
           hold on
           grid on

           dprofile = TU.tdepthp;
           pprofile = TU.Pprofile;

           plot(pprofile, dprofile(end:-1:1), '--
o', 'MarkerFaceColor', 'b')

           set(gca, 'XLim', [round(min(pprofile*0.95))
 round(max(pprofile*1.05))], ...
                    'YLim', [round(min(dprofile*0.95))
 round(max(dprofile*1.05))], ...
                    'FontName', 'Segoe UI', 'YDir', 'Reverse')
```

```
            xlabel(gca, 'Pressure (kPa)')
            ylabel(gca, 'TDV (m)')
            title(gca, 'Pressure profile in the tubing')
            legend('Pressure drop along the
 tubing', 'Location', 'southwest');
            hold off
        end

        function PlotTprofile(TU)
            %PlotTprofile. Plot temperature - depth
            % Plotting the temperature profile of the well along its
 depth

            if ~TU.status
                error('TubingObj:NotEnoughArguments', ...
                      'The tubing element must be properly defined')
            end

            figure
            axes
            hold on
            grid on

            dprofile = TU.tdepthp;
            tprofile = TU.Tprofile;

            plot(tprofile, dprofile(end:-1:1), '--
o', 'MarkerFaceColor', 'b')

            set(gca, 'XLim', [round(min(tprofile*0.95))
 round(max(tprofile*1.05))], ...
                     'YLim', [round(min(dprofile*0.95))
 round(max(dprofile*1.05))], ...
                     'FontName', 'Segoe UI', 'YDir', 'Reverse')
            xlabel(gca, 'Temperature (C)')
            ylabel(gca, 'TDV (m)')
            title(gca, 'Temperature profile in the tubing')
            legend('Tempeture gradient along the
 tubing', 'Location', 'southwest');
            hold off
        end

        function PlotPTprofile(TU)
            %PlotPTprofile. Plot temperature - depth
            % Plotting the temperature profile of the well along its
 depth

            if ~TU.status
                error('TubingObj:NotEnoughArguments', ...
                      'The tubing element must be properly defined')
            end

            figure
```

```
            dprofile = TU.tdepthp;
            tprofile = TU.Tprofile;
            pprofile = TU.Pprofile;

            cat =
 axes('XAxisLocation', 'bottom', 'YAxisLocation', 'left', ...
                        'Box', 'on');
            psp = get(cat, 'Position');
            cap = axes('Position', psp, 'XAxisLocation', 'top', ...

  'YAxisLocation', 'right', 'Box', 'on', 'Color','none');

            xlabel('Temperature (C)', 'Parent', cat)
            ylabel('TDV (m)', 'Parent', cat)
            xlabel('Pressure (kPa)', 'Parent', cap)
            ylabel('TDV (m)', 'Parent', cap)

            set(cat, 'XLim', [round(min(tprofile*0.95))
 round(max(tprofile*1.05))], ...
                        'YLim', [round(min(dprofile*0.95))
 round(max(dprofile*1.05))], ...
                        'FontName', 'Segoe UI', 'YDir', 'Reverse')
            set(cap, 'XLim', [round(min(pprofile*0.95))
 round(max(pprofile*1.05))], ...
                        'YLim', [round(min(dprofile*0.95))
 round(max(dprofile*1.05))], ...
                        'FontName', 'Segoe UI', 'YDir', 'Reverse')
            line(pprofile, dprofile(end:-1:1), 'Parent',
 cap, 'Color', 'r', ...
                    'LineStyle','--')
            line(tprofile, dprofile(end:-1:1), 'Parent',
 cat, 'Color', 'b',...
                    'LineStyle','--')

            grid on
          end
     end
end
```

Published with MATLAB® R2015a