```
classdef FlowlineObj < handle
%FlowlineObj. Flowline Class
% FlowlineObj represents a flowline connecting different elements in
 the
% surface for a network. It allows calculation of pressure/temperature
% profile along the flowline.
%
%   Properties
%    - Qsc             : flowrate at standard conditions    [Sm3/d]
%    - Qi              : flowrate flowline inlet            [m3/d]
%    - Qo              : flowrate flowline outlet           [m3/d]
%    - Pi              : pressure flowline outlet           [kPa]
%    - Po              : pressure flowline inlet            [kPa]
%    - Ti              : temperature flowline inlet         [C]
%    - To              : temperature flowline outlet        [C]
%    - rhosc           : density at standard conditions     [kg/m3]
%    - tTinf           : flowline surounding temperature    [K]
%    - tln             : flowline length                    [m]
%    - tdi             : flowline internal diameter         [m]
%    - trg             : flowline roughness                 [m]
%    - tuc             : flowline global heat coefficient   [kW/m2.K]
%    - tin             : flowline inclination (0 = hztal)   [rad]
%    - tsl             : flowline segment length            [m]
%    - BOmodel         : black oil model
%    - CalculationType : {forward | backward}
%
%   Constant properties
%    - g               : gravity acceleration               [m/s2]
%
%   Dependent properties
%    - status          : programming property (to be used in further
 versions)
%    - Tprofile        : flowline temperature profile       [C]
%    - Pprofile        : flowline pressure profile          [kPa]
%    - tlengthp        : flowline length profile            [m]
%
%   Methods
%    - SolveFlowline
%        Depending on the calculation type, calculates P, T and Q on
 the
%        other flowline nozzle.
%    - PlotPprofile
%        Plotting pressure profile along the flowline length.
%    - PlotTprofile
%        Plotting temperature profile along the flowline length.
%    - PlotPTprofile
%        Plotting temperature and pressure profile along the flowline
%        length.
%
 ---------------------------------------------------------------------
% Development
%   By: Ruben Ensalzado
```

```
%       2016
%   Rev 00 160521 original release
%                 improvement in Tprofile and Pprofile calculations

    properties
        BOmodel
        Qsc
        rhosc
        Qi
        Qo
        Pi
        Po
        Ti
        To
        tTinf
        tln
        tdi
        trg
        tuc
        tin
        tsl
        CalculationType
    end

    properties (Constant)
        g = 9.81;
    end

    properties (Dependent)
        status
        Tprofile
        Pprofile
        tlengthp
    end

    methods (Hidden)
        % -- initialization method --

        function TU = FlowlineObj
            TU.CalculationType = 'forward';
            TU.tsl = 2;
            TU.tin = 0;
        end
    end

    methods
        % -- error verification methods --

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

```
                    '- backward            '];

                error('FlowlineObj: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('FlowlineObj:BadArgument', ...
                      'The fluid model object has to be class BOObj')
            end
        end

        function set.Qsc(TU, uQsc)
            if size(uQsc, 1) == 3 && size(uQsc, 2) == 1
                TU.Qsc = uQsc;
            else
                error('FlowlineObj:BadArgument', ...
                      'Qsc must be a 3x1 double array.')
            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.tTinf);
            req(6) = isempty(TU.tdi);
            req(7) = isempty(TU.trg);
            req(8) = isempty(TU.tuc);

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

        function Tprofile = get.Tprofile(TU)
            %FlowlineTemperatureProfile
            %Units
```

```
            % T0 : C

            switch lower(TU.CalculationType)
                case 'forward'
                    T0 = TU.Ti + 273.15;
                case 'backward'
                    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)
            %FlowlinePressureProfile.
            %Units
            % P0       : kPa
            % Pprofile : kPa

            switch lower(TU.CalculationType)
                case 'forward'
                    P0 = TU.Pi;
                case 'backward'
                    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 tlengthp = get.tlengthp(TU)
            %FlowlinelengthProfile.
            %Units
            % tln : m
            % tsl : m

            sn = round(TU.tln/TU.tsl) + 1;
            tlengthp = linspace(0, TU.tln, sn + 1)';
        end
    end

    methods
        function SolveFlowline(TU)
            switch lower(TU.CalculationType)
                case 'forward'
                    TU.Po = TU.Pprofile(end);
                    TU.To = TU.Tprofile(end) - 273.15;
                case 'backward'
                    TU.Pi = TU.Pprofile(1);
```

```
                    TU.Ti = TU.Tprofile(1) - 273.15;
                end

                TU.QCalculation
            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 : flowline internal diameter
                % uc : flowline 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 'forward'

                    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 'backward'

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

            % ted : flowline 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(pi/2 - 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 'forward'

                    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 'backward'

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

            switch lower(TU.CalculationType)
                case 'forward'
                    Q = TU.Qi;
                case 'backward'
                    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.tlengthp;             % 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 type
        case 'Tprofile'
            props = zeros(n + 1, 3);
            props(:, 1) = emurho(TU)*16.0186;    % kg/m3
            props(:, 2) = emuCp(TU)*4.2216;      % kJ/kg.K
            props(:, 3) = TU.tTinf + 273.15;     % 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 'forward'
            qo = TU.Qi;
        case 'backward'
            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 'forward'
                    qo = TU.Qi;
                case 'backward'
                    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

        function nT = conversionT(~, oT, type)
            %conversionT. Temperature
            switch lower(type)
                case 'c'
                    nT = (oT - 32)/1.8;
                case 'f'
                    nT = oT*1.8 + 32;
            end
        end

        function nP = conversionP(~, oP, type)
            %conversationP. Pressure
            switch lower(type)
                case 'psi'
                    nP = oP*14.7/101.325;
                case 'kpa'
                    nP = oP*101.325/14.7;
            end
        end


        function QCalculation(TU)
            %Qcalculation. Flowrate at actual conditions
            %Units
            % P: kPa
            % T: C
```

```
            if isempty(TU.Qsc)
                return
            end

            TU.BOmodel.P = conversionP(TU, TU.Po, 'psi');
            TU.BOmodel.T = conversionT(TU, TU.To, 'F');

            qs2r = TU.BOmodel.qsc2ac;
            qiac = qs2r*TU.Qsc;

            switch lower(TU.CalculationType)
                case 'forward'
                    TU.Qo = sum(qiac);
                case 'backward'
                    TU.Qi = sum(qiac);
            end
        end

    end

    methods
        % -- information and plotting methods --

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

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

            figure
            axes
            hold on
            grid on

            dprofile = TU.tlengthp;
            pprofile = TU.Pprofile;

            plot(dprofile, pprofile, '--o', 'MarkerFaceColor', 'b')

            set(gca, 'YLim', [round(min(pprofile*0.95))
round(max(pprofile*1.05))], ...
                     'XLim', [round(min(dprofile*0.95))
round(max(dprofile*1.05))], ...
                     'FontName', 'Segoe UI')
            xlabel(gca, 'Flowline length (m)')
            ylabel(gca, 'Pressure (kPa)')

            title(gca, 'Pressure profile in the flowline')
            legend('Pressure drop along the
flowline', 'Location', 'southwest');
```

```
            hold off
        end

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

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

            figure
            axes
            hold on
            grid on

            dprofile = TU.tlengthp;
            tprofile = TU.Tprofile;

            plot(dprofile, tprofile, '--o', 'MarkerFaceColor', 'b')

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

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

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

            figure

            dprofile = TU.tlengthp;
            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');

            ylabel('Temperature (K)', 'Parent', cat)
            xlabel('Flowline length (m)', 'Parent', cat)
            ylabel('Pressure (kPa)', 'Parent', cap)
            xlabel('Flowline length (m)', 'Parent', cap)

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

            grid on
        end
    end
end
```

Published with MATLAB® R2015a