```
classdef NodeObj < handle
%NodeObj. Node Class
% NodeObj represents a node connecting different branches or nodes. In
 the
% node, the production of different branches and nodes commingle.
%
%   Properties
%    - BOmodel     : black oil model
%    - Branches    : cell array including branches' objects
%    - Nodes       : cell array including nodes' objects
%    - P           : pressure at the node                    [kPa]
%    - T           : temperature at the node                 [C]
%
%   Dependent properties
%    - Qsc         : flowrate at standard conditions         [Sm3/d]
%    - Qac         : flowrate at actual conditions           [m3/d]
%    - BranchCount : number of banches in the node
%    - NodeCount   : number of nodes in the object
%
%   Methods
%    - SolveBranches
%        Solving all branches included in the node, using the secant
%        algorithm.
%    - SolveNodes
%        Solving all nodes included in the node, using the secant
%        algorithm.
%
 ------------------------------------------------------------------------
% Development
%   By: Ruben Ensalzado
%       2016
%   Rev 00 160604 original release

    properties
        Branches
        Nodes
        P
        dP
        Pj
        Fj
        Jj
    end

    properties (SetAccess = protected)
        T
        BOmodel
    end

    properties (Dependent)
        Qsc
        Qac
        BranchCount
```

```
    NodeCount
end

properties (Hidden)
    Pe
end

properties (Hidden, SetAccess = immutable)
    fobj
    fpobj
    INmodel
    Tref
end

methods (Hidden)
    % -- initialization method --

    function ND = NodeObj
        ND.Branches = [];
        ND.P = 6000;
        ND.dP = 20;

        ND.fobj = @(Pd, Pc)(Pc - Pd)/(Pd + (Pd == 0));
        ND.fpobj = @(Ps, fPdP, fP)(1/Ps)*(fPdP - fP)/ND.dP;
        ND.INmodel = InjectionObj;
        ND.BOmodel = BOObj;
        ND.BOmodel.Tbosc = 851 - 460;
        ND.BOmodel.Tbgsc = 196.47 - 460;
        ND.BOmodel.GOR = 130;
        ND.Tref = 50;
    end
end

methods
    % -- error verification methods --

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

methods
    % -- dependent properties --

    function Qsc = get.Qsc(ND)
        %Qsc. Flowrate at standard conditions
        %Units
        % Qsc : Sm3/d
```

```
            n = ND.BranchCount;

            if ~n
                Qsc = [];
                return
            end

            Qsc = zeros(3, 1);

            for i = 1:n
                Qsc = Qsc + ND.Branches{i}{2}.Qsc;
            end
        end

        function Qac = get.Qac(ND)
            %Qac. Flowrate at standard conditions
            %Units
            % Qac : m3/d

            n = ND.BranchCount;

            if ~n
                Qac = [];
                return
            end

            for i = 1:n
                Qac = Qac + ND.Branches{i, end}.Qo;
            end
        end

        function BranchCount = get.BranchCount(ND)
            %BranchCount. Counting the number of branches

            if isempty(ND.Branches)
                BranchCount = 0;
            else
                BranchCount = size(ND.Branches, 2);
            end
        end

        function NodeCount = get.NodeCount(ND)
            %NodeCount. Counting the number of nodes

            if isempty(ND.Nodes)
                NodeCount = 0;
            else
                NodeCount = size(ND.Nodes, 2);
            end
        end

    end

    methods
```

```
function SolveBranches(ND)
    %SolveBranches
    % Solving all branches contained in the node.

    lpmax = 50;
    n = ND.BranchCount;

    Pi = zeros(n, lpmax);
    fi = zeros(n, lpmax);

    %first point
    Pna = ones(n, 1)*ND.P*0.8;
    dPna = ObjFcn(ND, Pna, n, 'branch');

    %second point
    Pnb = ones(n, 1)*ND.P*1.3;
    dPnb = ObjFcn(ND, Pnb, n, 'branch');

    Pnc = Pna;
    Pi(:, 1) = Pnc;

    for i = 2:lpmax
        m = (dPna - dPnb)./(Pna - Pnb);
        Pnc = Pnb - dPnb./m;

        dPnc = ObjFcn(ND, Pnc, n, 'branch');

        for j = 1:n
            if dPna(j)*dPnc(j) > 0
                dPna(j) = dPnc(j);
                Pna(j) = Pnc(j);
            else
                dPnb(j) = dPnc(j);
                Pnb(j) = Pnc(j);
            end
        end

        Pi(:, i) = Pnc;
        fi(:, i-1) = dPnc;

        if max(abs(dPnc)) < 1e-2
            break
        end
    end

    ND.Pj = Pi(:, 1:i);
    ND.Fj = fi(:, 1:i-1);

    ND.BOmodel.P = conversionP(ND, ND.P, 'psi');
    ND.AdiabaticMixing('branch')
    ND.ViscosityAdjustment('branch')
end

function SolveNodes(ND)
```

```
            %SolveNodes
            % Solving all NodeObj included in the current object.

            lpmax = 50;
            n = ND.NodeCount;

            Pi = zeros(n, lpmax);
            fi = zeros(n, lpmax);

            %first point
            Pna = ones(n, 1)*ND.P*0.8;
            dPna = ObjFcn(ND, Pna, n, 'node');

            %second point
            Pnb = ones(n, 1)*ND.P*1.3;
            dPnb = ObjFcn(ND, Pnb, n, 'node');

            for i = 2:lpmax
                m = (dPna - dPnb)./(Pna - Pnb);
                Pnc = Pnb - dPnb./m;

                dPnc = ObjFcn(ND, Pnc, n, 'node');

                for j = 1:n
                    if dPna(j)*dPnc(j) > 0
                        dPna(j) = dPnc(j);
                        Pna(j) = Pnc(j);
                    else
                        dPnb(j) = dPnc(j);
                        Pnb(j) = Pnc(j);
                    end
                end

                Pi(:, i) = Pnc;
                fi(:, i-1) = dPnc;

                if max(abs(dPnc)) < 1e-2
                    break
                end
            end

            ND.Pj = Pi(:, 1:i);
            ND.Fj = fi(:, 1:i-1);

            ND.BOmodel.P = conversionP(ND, ND.P, 'psi');
            ND.AdiabaticMixing('node')
            ND.ViscosityAdjustment('node')
        end
    end

    %(Hidden)
    methods
        function SolveBranch(ND, i)
            %SolveBranch
```

```
            % Solve branches. A branch is composed by two elements: a
            % SingleWellObj and a Flowline.

            obj = ND.Branches{i};

            obj{1}.Pwh = ND.Pe;
            obj{1}.SolveSingleWell

            obj{2}.BOmodel = obj{1}.BOmodel;
            obj{2}.Qsc = reshape(obj{1}.flowrates(end, 1:3), [3 1]);
            obj{2}.rhosc = conversionrho(ND, reshape(obj{1}.props(end,
4:6), [3 1]), 'kg/m3');
            obj{2}.Qi = obj{1}.flowrates(end, 7);
            obj{2}.Pi = obj{1}.props(end, 1);
            obj{2}.Ti = obj{1}.props(end, 2);
            obj{2}.SolveFlowline
        end

        function SolveNode(ND, i)
            %SolveNode
            % Solve nodes. A node is composed by two elements: a
NodeObj
            % and a Flowline

            obj = ND.Nodes{i};

            obj{1}.P = ND.Pe;
            obj{1}.SolveBranches

            obj{2}.BOmodel = obj{1}.BOmodel;
            obj{2}.BOmodel.P = conversionP(ND, obj{1}.P, 'psi');
            obj{2}.BOmodel.T = conversionT(ND, obj{1}.T, 'F');
            obj{2}.Qsc = obj{1}.Qsc;
            obj{2}.rhosc = WeightDensity(ND, obj{1});

            qs2r = obj{2}.BOmodel.qsc2ac;

            qac = qs2r*obj{2}.Qsc;
            obj{2}.Qi = sum(qac(2:3));
            obj{2}.Pi = obj{1}.P;
            obj{2}.Ti = obj{1}.T;
            obj{2}.SolveFlowline
        end

        function Fv = ObjFcn(ND, P, n, type)
            %ObjFcn
            % Objective function to solve branches or nodes using
secant
            % algorithm.

            Fv = zeros(n, 1);
            fval = zeros(n, 1);

            switch lower(type)
```

```
                    case 'branch'
                        for i = 1:n
                            ND.Pe = P(i);
                            ND.SolveBranch(i)
                            fval(i, 1) = ND.Branches{i}{2}.Po;
                            Fv(i) = ND.fobj(ND.P, fval(i, 1));
                        end
                    case 'node'
                        for i = 1:n
                            ND.Pe = P(i);
                            ND.SolveNode(i)
                            fval(i, 1) = ND.Nodes{i}{2}.Po;
                            Fv(i) = ND.fobj(ND.P, fval(i, 1));
                        end
                end
            end

            function AdiabaticMixing(ND, type)
                %AdiabaticMixing
                % Adiabatic mixing of streams commingling production from
                % different branches.

                switch lower(type)
                    case 'branch'
                        n = ND.BranchCount;
                    case 'node'
                        n = ND.NodeCount;
                end

                mb = zeros(3, n);
                qb = zeros(3, n);
                eb = zeros(n, 1);
                rhosc = zeros(3, n);

                for i = 1:n
                    switch lower(type)
                        case 'branch'
                            obj = ND.Branches{i};
                        case 'node'
                            obj = ND.Nodes{i};
                    end

                    obj{2}.BOmodel.T = conversionT(ND, obj{2}.To, 'F');
                    qb(:, i) = obj{2}.Qsc;
                    qs2r = obj{2}.BOmodel.qsc2ac;
                    qaci = qs2r*qb(:, i)/(3600*24);
                    rhoi = conversionrho(ND, [obj{2}.BOmodel.grho;...
                                              obj{2}.BOmodel.orho;...
                                              obj{2}.BOmodel.wrho], 'kg/
m3');
                    mi = qaci.*rhoi;
                    mb(:, i) = mi;
                    Cpi = conversionCp(ND, [obj{2}.BOmodel.gCp;...
                                            obj{2}.BOmodel.oCp;...
```

```
                                        obj{2}.BOmodel.wCp], 'kj/
kg.k');
                eb(i) = sum(obj{2}.To*mi.*Cpi);
                rhosc(:, i) = obj{2}.rhosc;
            end

            ND.BOmodel.gSG = sum(rhosc(1, :).*mb(1, :))/(sum(mb(1, :))
 + all(~mb(1, :)))/1.205;
            ND.BOmodel.oSG = sum(rhosc(2, :).*mb(2, :))/(sum(mb(2, :))
 + all(~mb(2, :)))/1000;
            ND.BOmodel.wSG = sum(rhosc(3, :).*mb(3, :))/(sum(mb(3, :))
 + all(~mb(3, :)))/1000;

            Tm = ND.Tref;

            for i = 1:50
                [FT, DT] = MixingTemperature(Tm);
                Tm = Tm -  FT/DT;

                if abs(FT) < 1e-2
                    break
                end
            end

            ND.T = Tm;
            ND.BOmodel.T = conversionT(ND, Tm, 'F');

            function [FT, DT] = MixingTemperature(T0)
                %MixingTemperature.
                % Objective function function for the adiabatic mixing
 of
                % streams from branches.

                dT = 1;

                ND.BOmodel.T = conversionT(ND, T0, 'F');
                Cpm = conversionCp(ND, [ND.BOmodel.gCp; ...
                                        ND.BOmodel.oCp; ...
                                        ND.BOmodel.wCp], 'kj/kg.k');
                FT = T0 - sum(eb)/sum(sum(mb, 2).*Cpm);

                ND.BOmodel.T = conversionT(ND, T0 + dT, 'F');
                Cpm = conversionCp(ND, [ND.BOmodel.gCp; ...
                                        ND.BOmodel.oCp;
                                        ND.BOmodel.wCp], 'kj/kg.k');
                FdT = (T0 + dT) - sum(eb)/sum(sum(mb, 2).*Cpm);
                DT = (FdT - FT)/dT;
            end
        end

        function ViscosityAdjustment(ND, type)
            %ViscosityAdjustment.
            % Calculation to set the resulting viscosity at the node.
```

```
switch lower(type)
    case 'branch'
        n = ND.BranchCount;
    case 'node'
        n = ND.NodeCount;
end

mrT = zeros(n, 2);
vmu = zeros(n, 2);
qsc = zeros(3, n);
rhosc = zeros(3, n);

for i = 1:n
    switch lower(type)
        case 'branch'
            obj = ND.Branches{i};
        case 'node'
            obj = ND.Nodes{i};
    end

    mrT(i, 1) = obj{2}.To;
    obj{2}.BOmodel.T  = conversionT(ND, mrT(i, 1), 'F');
    vmu(i, 1) = obj{2}.BOmodel.omu;

    mrT(i, 2) = obj{2}.To + 20;
    obj{2}.BOmodel.T  = conversionT(ND, mrT(i, 2), 'F');
    vmu(i, 2) = obj{2}.BOmodel.omu;

    qsc(:, i) = obj{2}.Qsc;
    rhosc(:, i)= obj{2}.rhosc;
end

qo = qsc(2, 1);
orho = rhosc(2, 1);
ovmu = vmu(1, :);
omrT = mrT(1, :);

obj = ND.INmodel;

if n > 1
    for i = 1:n-1
        obj.Qo = qo;
        obj.Qd = qsc(2, i + 1);

        obj.orho = orho;
        obj.drho = rhosc(2, i + 1);

        obj.mrmu = [ovmu; vmu(i + 1, :)];
        obj.mrT = [omrT; mrT(i + 1, :)];

        obj.Tb = ND.T;
        obj.BlendingReference

        qo = sum(qsc(2, i:i+1));
```

```
                    orho = obj.brho;
                    ovmu = reshape(obj.bmrmu([1 3]), [1 2]);
                    omrT = reshape(obj.bmrT([1 3]), [1 2]);
                end
            else
                obj.Qo = qo;
                obj.Qd = 0;
                obj.orho = orho;
                obj.drho = 0;

                obj.mrmu = ovmu;
                obj.mrT = omrT;

                obj.Tb = ND.T;
                obj.BlendingReference
            end

                bmrmu = obj.bmrmu;
                bmrT = obj.bmrT;
                ND.BOmodel.TuneViscosity(bmrmu, conversionT(ND,
bmrT, 'F'));
        end

        function rhosc = WeightDensity(~, obj)

            n = obj.BranchCount;
            qj = zeros(3, n);
            rj = zeros(3, n);

            for j = 1:n
                qj(:, j) = obj.Branches{j}{2}.Qsc;
                rj(:, j) = obj.Branches{j}{2}.rhosc;
            end

            rhosc = sum(qj.*rj.^2, 2)./(sum(qj.*rj, 2) + sum(~qj, 2));
        end
    end

    methods (Hidden)
        % -- unit conversion functions --

        % programmer notes:
        %  integrating these functions as methods, instead of list
them
        %  as separate fuunctions is 1.0% - 1.5% faster.

        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 nrho = conversionrho(~, orho, type)
            %conversionrho. Density
            switch lower(type)
                case 'kg/m3'
                    nrho = orho*16.0186;
                case 'lb/ft3'
                    nrho = orho/16.0186;
            end
        end

        function nCp = conversionCp(~, oCp, type)
                    %conversionrho. Mass
            switch lower(type)
                case 'kj/kg.k'
                    nCp = oCp*4.2216;
                case 'btu/lb.r'
                    nCp = oCp/4.2216;
            end
        end
    end
end
```

Published with MATLAB® R2015a