```
classdef SingleWellObj < handle
%SingleWellObj. Single Well Class
% SingleWellObj represents a single well model. This object may
 include
% completions, tubings, esps and injection points.
%
%   Properties
%    - BOmodel    : black oil model
%    - Pwh        : wellhead pressure                    [kPa]
%    - items      : items conforming the system
%    - flowrates  : matrix including items's flowrates  [Sm3/d, m3/d]
%    - props      : matrix including fluid's properties
%    - WC         : water cut @ sc                       [%]
%    - mrT        : reference matrix temperature         [C]
%    - mrmu       : reference matrix viscosity           [cP or cSt]
%
%   Dependent properties
%    - status     : programming property (to be used in further
 versions)
%    - ItemsCount : number of system's items
%
%   Methods
%    - SolveSingleWell
%        Solving the single well structure, calculating all flowrates
 and
%        properties for a given wellhead pressure.
%
%   Remark on properties
%    <flowrates>  Matrix (n x 7) including items flowrates, where n is
 the
%                 number of items, and item number 1 is always a
 completion
%                 object. Flowrates are as follows (column index is
%                 enclosed on parathesis):
%                 (1) Qgsc : gas flowrate   @standard conditions [Sm3/
d]
%                 (2) Qosc : oil flowrate   @standard conditions [Sm3/
d]
%                 (3) Qwsc : water flowrate @standard conditions [Sm3/
d]
%                 (4) Qg   : gas flowrate   @local conditions    [m3/
d]
%                 (5) Qo   : oil flowrate   @local conditions    [m3/
d]
%                 (6) Qw   : water flowrate @local conditions    [m3/
d]
%                 (7) Qt   : total flowrate (gas + water + oil)  [m3/
d]
%
%    <props>      Matrix (n x 10) including items properties, where n
 is
%                 the number of items, and item number 1 is always a
```

```
%                completion object. Flowrates are as follows (column
 index
%                is enclosed on parathesis):
%                ( 1) P      : pressure
 [kPa]
%                ( 2) T      : temperature                          [K]
%                ( 3) WC     : watercut @local conditions           [%]
%                ( 4) grhosc : gas density   @standard conditions
 [kg/m3]
%                ( 5) orhosc : oil density   @standard conditions
 [kg/m3]
%                ( 6) wrhosc : water density @standard conditions
 [kg/m3]
%                ( 7) grho   : gas density   @local conditions
 [kg/m3]
%                ( 8) orho   : oil density   @local conditions
 [kg/m3]
%                ( 9) wrho   : water density @local conditions
 [kg/m3]
%                (10) emu    : effective emulsion viscosity
 [cP]
%
 ------------------------------------------------------------------------
% Development
%   By: Ruben Ensalzado
%       2016
%   Rev 00 160117 original release

% Developer notes:
%  Gas flowrate is not included in the total flowrate at local
 conditions;
%  the current objects do not work with gas phase. When adapted for
 that,
%  locate the lines:
%     'SW.flowrates(i, 7) = sum(SW.flowrates(i, 4:6));'
%  and enable them

    properties
        BOmodel
        Pwh
        items
        WC
        mrT
        mrmu
    end

    properties (SetAccess = protected)
        flowrates
        props
    end

    properties (Dependent)
        status
        ItemsCount
```

```
    end

    properties (Hidden)
        Qsci
    end

    properties (Hidden, SetAccess = immutable)
        Pwherf
    end

    events
        NoConvergenceSign
        NoConvergenceIter
    end

    methods
        % -- initialization method --

        function SW = SingleWellObj
            SW.Pwh = 60*100;
            SW.items = {};
            SW.Qsci = 2000;
            SW.props = zeros(1, 8);
            SW.flowrates = zeros(1, 7);

            SW.WC = 0;
            SW.mrT = [];
            SW.mrmu = [];
            SW.Pwherf = @(Pc, Pd) (Pd - Pc)/(Pd + (Pd == 0));
            addlistener(SW, 'NoConvergenceSign',
@SingleWellObj.NoConvergenceFcn);
            addlistener(SW, 'NoConvergenceIter',
@SingleWellObj.NoConvergenceFcn);
        end
    end

    methods
        % -- error verification methods --

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

    end

    methods
        function status = get.status(SW)
            %Status. Independent property verification
```

```
            req = zeros(9, 1);
            req(1) = ~iscell(SW.items);

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

        function ItemsCount = get.ItemsCount(SW)
            ItemsCount = length(SW.items);
        end
    end

    methods (Hidden)
        % -- private methods --

        function SolveObject(SW, Obj, i)
            %SolveObject.
            % Solve every object accoring to its type, and create the
            % properties and flow rate matrices.

            % programmer notes:
            % qisc: q item, standard conditions
            % qiac: q item, actual conditions

            switch lower(class(Obj))
                case 'vertcompletionobj'
                    % Vertical completion object

                    switch lower(Obj.IPRType)
                        case 'well pi'
                            switch lower(Obj.WellPIType)
                                case 'gas'
                                    Obj.Qsc = SW.Qsci;
                                case 'oil'
                                    Obj.Qsc = SW.Qsci;
                            end
                        case 'jones'
                            switch lower(Obj.JonesType)
                                case 'gas'
                                    Obj.Qsc = SW.Qsci;
                                case 'oil'
                                    Obj.Qsc = SW.Qsci;
                            end
                        otherwise
                            Obj.Qsc = SW.Qsci;
                    end

                    Obj.SolveCompletion

                    if ~isempty(SW.mrT)
                        bmrT = SW.mrT;
```

```
                        bmrmu = SW.mrmu;
                        SW.BOmodel.TuneViscosity(bmrmu,
conversionT(SW, bmrT, 'F'));
                    end

                    SW.BOmodel.P = 14.7;
                    SW.BOmodel.T = 60;
                    SW.BOmodel.WC = SW.WC;

                    qisc = [Obj.Qsc*SW.BOmodel.GOR; ...
                            Obj.Qsc; ...
                            Obj.Qsc*SW.BOmodel.WC/(100 -
SW.BOmodel.WC)];
                    risc = [SW.BOmodel.grho; ...
                            SW.BOmodel.orho; ...
                            SW.BOmodel.wrho];

                    SW.BOmodel.P = conversionP(SW, Obj.Pwf, 'psi');
                    SW.BOmodel.T = conversionT(SW, Obj.Twf, 'F');

                    qs2r = SW.BOmodel.qsc2ac;
                    qiac = qs2r*qisc;

                    SW.flowrates(1, 1) = qisc(1);
                    SW.flowrates(1, 2) = qisc(2);
                    SW.flowrates(1, 3) = qisc(3);
                    SW.flowrates(1, 4) = qiac(1);
                    SW.flowrates(1, 5) = qiac(2);
                    SW.flowrates(1, 6) = qiac(3);
                    SW.flowrates(1, 7) = sum(SW.flowrates(1, 5:6));
                    %SW.flowrates(1, 7) = sum(SW.flowrates(1, 4:6));

                    rs2r = SW.BOmodel.rhosc2ac;
                    riac = rs2r*risc;

                    SW.props(1, 1) = Obj.Pwf;
                    SW.props(1, 2) = Obj.Twf;
                    SW.props(1, 3) = SW.BOmodel.WC;
                    SW.props(1, 4) = risc(1);
                    SW.props(1, 5) = risc(2);
                    SW.props(1, 6) = risc(3);
                    SW.props(1, 7) = riac(1);
                    SW.props(1, 8) = riac(2);
                    SW.props(1, 9) = riac(3);
                    SW.props(1, 10) = SW.BOmodel.emu;

                case 'injectionobj'
                    % Injection object

                    Obj.Qo = SW.flowrates(i - 1, 2);
                    Obj.Qw = SW.flowrates(i - 1, 3);

                    Obj.Tb = SW.props(i - 1, 2);
```

```
                    Obj.orho = conversionrho(SW, SW.props(i - 1,
5), 'kg/m3');
                    Obj.BlendingReference

                    bmrmu = Obj.bmrmu;
                    bmrT = Obj.bmrT;

                    SW.BOmodel.TuneViscosity(bmrmu, conversionT(SW,
bmrT, 'F'));
                    SW.BOmodel.P = conversionP(SW, SW.props(i - 1,
1), 'psi');
                    SW.BOmodel.T = conversionT(SW, SW.props(i - 1,
2), 'F');

                    qisc = [SW.flowrates(i - 1, 1); ...
                            Obj.Qo + Obj.Qd; ...
                            SW.flowrates(i - 1, 3)];
                    qs2r = SW.BOmodel.qsc2ac;
                    qiac = qs2r*qisc;

                    SW.flowrates(i, 1) = qisc(1);
                    SW.flowrates(i, 2) = qisc(2);
                    SW.flowrates(i, 3) = qisc(3);
                    SW.flowrates(i, 4) = qiac(1);
                    SW.flowrates(i, 5) = qiac(2);
                    SW.flowrates(i, 6) = qiac(3);
                    SW.flowrates(i, 7) = sum(SW.flowrates(i, 5:6));
                    %SW.flowrates(i, 7) = sum(SW.flowrates(i, 4:6));

                    SW.BOmodel.WC = 100*qiac(3)/(qiac(2) + qiac(3));
                    risc = [SW.props(i - 1, 4); ...
                            conversionrho(SW, Obj.brho, 'lb/ft3'); ...
                            SW.props(i - 1, 6)];
                    rs2r = SW.BOmodel.rhosc2ac;
                    riac = rs2r*risc;

                    SW.props(i, 1) = SW.props(i - 1, 1);
                    SW.props(i, 2) = SW.props(i - 1, 2);
                    SW.props(i, 3) = SW.BOmodel.WC;
                    SW.props(i, 4) = risc(1);
                    SW.props(i, 5) = risc(2);
                    SW.props(i, 6) = risc(3);
                    SW.props(i, 7) = riac(1);
                    SW.props(i, 8) = riac(2);
                    SW.props(i, 9) = riac(3);
                    SW.props(i, 10) = SW.BOmodel.emu;

                case 'espobj'
                    % ESP object

                    qo = SW.flowrates(i - 1, 5);
                    qw = SW.flowrates(i - 1, 6);
                    qt = qo + qw;
```

```
                    Pin = SW.props(i - 1, 1);
                    Tin = SW.props(i - 1, 2);
                    orho = conversionrho(SW, SW.props(i - 1, 8), 'kg/
m3');
                    wrho = conversionrho(SW, SW.props(i - 1, 9), 'kg/
m3');

                    mrho = (qo*orho + qw*wrho)/qt;
                    mSG = mrho/1e3;

                    Obj.vnu = SW.props(i - 1, 10);
                    Obj.Q = conversionF(SW, qt, 'm3/d - m3/h');
                    Obj.ViscosityAdjustment
                    Obj.FrequencyAdjustment

                    if any(Obj.CurveHQnc(:, 2) > Obj.Q)

                        Pot = Pin + Obj.H*mSG*9.81;

                        SW.BOmodel.P = conversionP(SW, Pot, 'psi');
                        SW.BOmodel.T = conversionT(SW, Tin, 'F');

                        qisc = [SW.flowrates(i - 1, 1); ...
                                SW.flowrates(i - 1, 2); ...
                                SW.flowrates(i - 1, 3)];
                        qs2r = SW.BOmodel.qsc2ac;
                        qiac = qs2r*qisc;

                        SW.flowrates(i, 1) = qisc(1);
                        SW.flowrates(i, 2) = qisc(2);
                        SW.flowrates(i, 3) = qisc(3);
                        SW.flowrates(i, 4) = qiac(1);
                        SW.flowrates(i, 5) = qiac(2);
                        SW.flowrates(i, 6) = qiac(3);
                        SW.flowrates(i, 7) = sum(SW.flowrates(i,
5:6));
                        %SW.flowrates(i, 7) = sum(SW.flowrates(i,
4:6));

                        SW.BOmodel.WC = 100*qiac(3)/(qiac(2) +
qiac(3));

                        risc = [SW.props(i - 1, 4); ...
                                SW.props(i - 1, 5); ...
                                SW.props(i - 1, 6)];
                        rs2r = SW.BOmodel.rhosc2ac;
                        riac = rs2r*risc;

                        SW.props(i, 1) = Pot;
                        SW.props(i, 2) = Tin;
                        SW.props(i, 3) = SW.BOmodel.WC;
                        SW.props(i, 4) = risc(1);
                        SW.props(i, 5) = risc(2);
                        SW.props(i, 6) = risc(3);
```

```
                        SW.props(i, 7) = riac(1);
                        SW.props(i, 8) = riac(2);
                        SW.props(i, 9) = riac(3);
                        SW.props(i, 10) = SW.BOmodel.emu;
                    else
                        %sustituir con flag
                        SW.props(i, 1) = -1;
                    end

                case 'tubingobj'
                    % Tubing object

                    SW.BOmodel.P = conversionP(SW, SW.props(i - 1,
1), 'psi');
                    SW.BOmodel.T = conversionT(SW, SW.props(i - 1,
2), 'F');
                    SW.BOmodel.WC = SW.props(i - 1, 3);

                    Obj.BOmodel = SW.BOmodel;
                    Obj.Pi = SW.props(i - 1, 1);
                    Obj.Ti = SW.props(i - 1, 2);
                    Obj.Qi = SW.flowrates(i - 1, 7);
                    Obj.SolveTubing

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

                    if SW.BOmodel.P > 0

                        qisc = [SW.flowrates(i - 1, 1); ...
                                SW.flowrates(i - 1, 2); ...
                                SW.flowrates(i - 1, 3)];
                        qs2r = SW.BOmodel.qsc2ac;
                        qiac = qs2r*qisc;

                        SW.flowrates(i, 1) = qisc(1);
                        SW.flowrates(i, 2) = qisc(2);
                        SW.flowrates(i, 3) = qisc(3);
                        SW.flowrates(i, 4) = qiac(1);
                        SW.flowrates(i, 5) = qiac(2);
                        SW.flowrates(i, 6) = qiac(3);
                        SW.flowrates(i, 7) = sum(SW.flowrates(i,
5:6));
                        %SW.flowrates(i, 7) = sum(SW.flowrates(i,
4:6));

                        SW.BOmodel.WC = 100*qiac(3)/(qiac(2) +
qiac(3));

                        risc = [SW.props(i - 1, 4); ...
                                SW.props(i - 1, 5); ...
                                SW.props(i - 1, 6)];
                        rs2r = SW.BOmodel.rhosc2ac;
                        riac = rs2r*risc;
```

```
                        SW.props(i, 1) = conversionP(SW,
SW.BOmodel.P, 'kPa');
                        SW.props(i, 2) = conversionT(SW,
SW.BOmodel.T, 'C');
                        SW.props(i, 3) = SW.BOmodel.WC;
                        SW.props(i, 4) = risc(1);
                        SW.props(i, 5) = risc(2);
                        SW.props(i, 6) = risc(3);
                        SW.props(i, 7) = riac(1);
                        SW.props(i, 8) = riac(2);
                        SW.props(i, 9) = riac(3);
                        SW.props(i, 10) = SW.BOmodel.emu;

                    elseif imag(SW.BOmodel.P) ~= 0 ||
isnan(SW.BOmodel.P)
                        %sustituir con flag!
                        SW.props(i, 1) = -1;
                    else
                        SW.props(i, 1) = conversionP(SW,
SW.BOmodel.P, 'kPa');
                    end
            end
        end

        function SolveObjects(SW)
            %SolveObjects.
            % Loop for solving all items in the single well.

            for i = 1:SW.ItemsCount
                CurrentObject = SW.items{i};
                SW.SolveObject(CurrentObject, i);

                %sustituir por flag!
                if any(SW.props(:, 1) < 0)
                    break
                end
            end
        end

    end

    methods
        % -- public methods --

        function SolveSingleWell(SW)
            % flowrates
            %  1     2     3     4     5     6     7
            %  Qgsc  Qosc  Qwsc  Qg    Qo    Qw    Qt
            %
            % props
            %  1      2      3      4      5      6      7      8
9      10
```

```
            %  P      T      WC     grhosc orhosc wrhosc grho   orho
    wrho   emu

            % programmer note:
            % The resolution method take two points for calculation in
    the
            % vertical completion: 0.95*AOF and 0.05*AOF.
            %
            % solving method: secant line

            SW.flowrates = zeros(SW.ItemsCount, 7);
            SW.props = zeros(SW.ItemsCount, 10);

            VC = SW.items{1};
            AOFP = VC.AOFP;

            %cond1 = 0;
            loopmax = 1e2;
            n = 100;

            Pwhfa = 0;
            Pwhfb = 0;
            Qscfa = 0;
            Qscfb = 0;

            vAOFP = linspace(0, AOFP, n);

            % first point
            for i = n - 1:-1:2
                SW.Qsci = vAOFP(i);
                SW.SolveObjects
                if any(SW.props(:, 1) <= 0)
                    SW.flowrates = zeros(SW.ItemsCount, 7);
                    SW.props = zeros(SW.ItemsCount, 10);
                else
                    Qscfa = SW.Qsci;
                    Pwhfa = SW.Pwherf(SW.props(end, 1), SW.Pwh);
                    break
                end
            end

            % second point
            for i = 2:1:n - 1
                SW.Qsci = vAOFP(i);
                SW.SolveObjects
                if any(SW.props(:, 1) <= 0)
                    SW.flowrates = zeros(SW.ItemsCount, 7);
                    SW.props = zeros(SW.ItemsCount, 10);
                else
                    Qscfb = SW.Qsci;
                    Pwhfb = SW.Pwherf(SW.props(end, 1), SW.Pwh);
                    if Pwhfb > 0
                        continue
                    end
```

```
        break
    end
end

if Pwhfa*Pwhfb > 0 || ~ Qscfa || ~ Qscfb
    notify(SW, 'NoConvergenceSign')
    return
end

Pwhfc = Pwhfa;
loop = 1;
dQ0 = abs(Qscfa - Qscfb)/2;

while abs(Pwhfc) > 1e-2
    m = (Pwhfa - Pwhfb)/(Qscfa - Qscfb);

    Qscfc = Qscfb - Pwhfb/m;
    SW.Qsci = Qscfc;
    SW.SolveObjects
    Pwhfc = SW.Pwherf(SW.props(end, 1), SW.Pwh);

    if abs(Pwhfc) < 1e-2 || dQ0 < 1
        break
    end

    if Pwhfa*Pwhfc > 0
        dQ1 = abs(Qscfc - Qscfb);
    else
        dQ1 = abs(Qscfc - Qscfa);
    end

    if dQ1 > dQ0
        Qscfc = (Qscfa + Qscfb)/2;
        SW.Qsci = Qscfc;
        SW.SolveObjects
        Pwhfc = SW.Pwherf(SW.props(end, 1), SW.Pwh);
    end

    if Pwhfa*Pwhfc > 0
        Pwhfa = Pwhfc;
        Qscfa = Qscfc;
    else
        Pwhfb = Pwhfc;
        Qscfb = Qscfc;
    end

    dQ0 = abs(Qscfb - Qscfa)/2;
    loop = loop + 1;
end

if loop > loopmax
    notify(SW, 'NoConvergenceIter')
    return
end
```

```
        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 nF = conversionF(~, oF, type)
            %conversionF. Volumetric flowrate
            switch lower(type)
                case 'm3/h'
                    nF = oF*0.1589873/24;
                case 'bpd'
                    nF = oF*24/0.1589873;
                case 'm3/h - m3/d'
                    nF = oF*24;
                case 'm3/d - m3/h'
                    nF = oF/24;
            end
        end
```

```
    end

    methods (Static)
        function NoConvergenceFcn(SWsrc, SWevent)
            switch SWevent.EventName
                case 'NoConvergenceSign'
                    state = ['SingleWellObj:NoConvergence
                ';...
                                'No sign change was found in the interval
 [0, AOFP].     ';...
                                'Try decreasing the interval for
 searching a sign change.'];
                case 'NoConvergenceIter'
                    state = ['SingleWellObj:NoConvergence
 ';...
                                'Maximum number of iterations
 reached.';...
                                'Try decreasing the Pwh value.        '];
            end
                fprintf(' %s \n %s \n %s \n', state(1, :),
 state(2, :), state(3, :));
                SWsrc.flowrates = zeros(SWsrc.ItemsCount, 7);
                SWsrc.props = zeros(SWsrc.ItemsCount, 10);
        end
    end
end
```

Published with MATLAB® R2015a