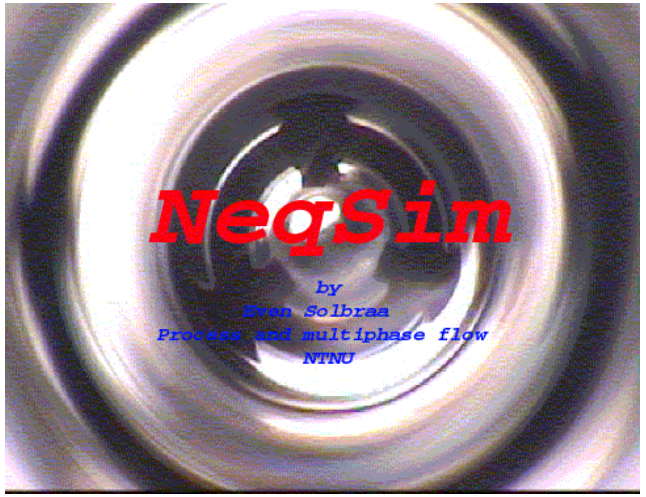
Getting started with NeqSim in Matlab

Even Solbraa/Pablo Dupoy



Outline

- Introduction to NeqSim for Matlab
- Setting up Matlab for first use of NeqSim
- Select thermodynamic model, create a fluid and add components
- Property calculations for single phase systems
  - Reading thermodynamic properties
  - Reading physical properties
- Equilibrium flash calculations
  - Flash calculations: TP, PH, PS, TV
  - Multiphase flash
  - Hydrocarbon bubble/dew point, phase envelope
  - Freezing in hydrocarbon systems (LNG)
- Thermodynamic calculations with water
  - Thermodynamic model selection
  - Water saturation, water content of a reservoir fluid
  - Water, ice and hydrate dew points
- Thermodynamic calculations with methanol and glycol (TEG, MEG)
  - Thermodynamic model selection
  - Phase equilibrium of gas, oil and glycol
  - Ice, solid glycol and complex freezing points
- Calculation of thermodynamic, physical and transport properties
  - Density, Enthalpies, Entropies, etc.
  - Viscosity, Conductivity
  - Interfacial tension
  - Solid adsorption
  - Diffusion coefficients
- Characterization of oils, PVT simulation and reservoir fluid tuning
  - TBP fractions and model selection
  - Defining and characterization of a fluid with a plus fractions
  - PVT simulation
  - PVT fluid tuning
- External tools integration
  - Generation of OLGA input fluids
  - Generation of HYSYS fluids
- Hydrate equilibrium
  - Hydrate equilibrium
  - Inhibitors and hydrate calculations
  - Hydrate equilibrium and salts
  - Top of line hydrate equilibrium
- Wax and asphaltene calculations
  - Prediction of wax equilibrium temperature
  - Prediction of asphaltene content
- Scale formation
  - Adding salt and ions to fluids
  - Calculation of scale formation and saturation ratio
- Trace components
  - Phase equilibrium of mercury in petroleum fluids
  - Phase equilibrium of sulfur components (H2S, COS; RSH, S8) in petroleum fluids
- Chemical equilibrium of petroleum fluids
  - Chemical equilibrium in petroleum reservoirs
  - Deposition, phase and chemical and equilibrium in petroleum transport and process (FeS, sulfur, HgS)
  - Equilibrium in combustion processes
- Thermodynamics of acid gas removal and injection
  - CO2 and H2S solubility in physical solvents
  - CO2 and H2S solubility in chemical solvent/alkanolamines
  - Hydrocarbon solubility in solvents
  - Thermodynamics of membranes for CO2/H2S removal
  - Thermodynamics of cryogenic CO2 removal (freezing/distillation)
  - Water solubility in CO2 and H2S and hydrocarbons
- Thermodynamics of dehydration processes
  - Thermodynamics of inhibition processes
  - Thermodynamics of absorption of water in glycol
  - Thermodynamics of water adsorption
- Fluid flow
  - One phase pipe flow
  - Two phase pipe flow
- Non-equilibrium calculations
  - Simulating two-phase non equilibrium processes
  - Simulating total liquid evaporation processes
  - Simulating nucleation and droplet growth
- Calculation of gas quality properties
  - GCV, WI and relative density calculations (ISO6969)
  - Gas density (AGA8) (ISO2132)
  - Calculating water dew point (ISO 5198)
  - Liquid density (Costald) (ISO5198)
- Process simulation
  - Creating streams
  - Process equipment
    - Valves
    - Separators and scrubbers
    - Compressors and pumps
    - Heat exchangers
    - Mixers
  - Recirculation streams
  - Pipelines (one-phase, two phase, multiphase)
  - Absorber and distillation columns
  - Working with process modules
- Process design
  - Pipeline design
  - Design of process units (static equipment, rotary equipment, etc.)
  - Designing a gas process plant
  - Estimation of plot space and weight
- Process economy
  - Cost estimation in NeqSim
  - Estimation of CAPEX and OPEX
- Advanced
  - Thermodynamic parameter fitting

Introduction to NeqSim for Matlab

- NeqSim – «Non equilibrium Simulator»[1)]
- A tool for process and flow assurance specialist work in Statoil:
  - Design of new advanced processes and equipment
  - A tool for operational support
  - A tool for taking R&D into business (solutions, patents, ..)
- Main modules:
  - Non-equilibrium and equilibrium thermodynamics
  - Physical properties
  - Chemical reaction equilibrium and kinetics
  - Fluid mechanics
  - Process simulation
  - Statistics and parameter fitting
  - Graphical user interface

Setting up Matlab for first use of NeqSim

- The NeqSim toolbox for matlab is available via GIT-hub:

  https://github.com/Statoil/neqsimmatlab

- To initialize calculations with NeqSim in Matlab, execute the expressions run the file:

```
pathNeqSim.m
```

- This will make NeqSim library and database available from Matlab
- Examples of Matlab scripts are found in the examples folder

  \matlab\example

Select thermodynamic model, create a fluid

- A fluid is created using the function:
  - *thermo('thermo model', temperature, pressure)*
  - A thermodynamic model need to be specified
  - Temperature and pressure are optional arguments
- Examples of valid thermodynamic models are: pr, srk, srk-mc, cpa, umr
- In the example script below, a fluid object with name fluid_1 is created, the thermodynamic model is SRK-EoS, and temperature and pressure are 10 bara and 273.15 Kelvin;

```
pressure = 10.0;            % pressure in bara
temperature = 273.15;       % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);
```

...and add components

- Valid components names can be obtained from the matlab function - componentNames
- Components are added to the fluid_1 object using the method
  - addComponent(component name, mole numbers)
  - Where valid component names are given in the table below, and mole numbers has unit mole/second

```
fluid_1.addComponent('methane', 1.0);          % adding 1 mole/second of methane
fluid_1.addComponent('propane', 1.0);          % adding 1 mole/second of propane
```

- Examples of valid component names:

water	CO2	n-pentane	3-m-C5	2-M-C6	224-TM-C5	3-M-C7	2-M-C8
MEG	methane	n-hexane	c-C5	23-DM-C5	113-TM-cy-C5	3-E-C6	ethylcyclohexane
TEG	ethane	benzene	iC5	11-DM-cy-C5	22-DM-C6	ethylbenzene	
DEG	propane	toluene	M-cy-C5	3-M-C6	E-cy-C5	m-Xylene	
methanol	n-butane	n-heptane	24-DM-C5	cis-13-DM-cy-C5	25-DM-C6	p-Xylene	
ethanol	i-butane	n-octane	223-TM-C4	trans-13-DM-cy-C5	24-DM-C6	o-Xylene	
nitrogen	i-pentane	c-propane	33-DM-C5	trans-12-DM-cy-C5	cis-13-DM-cy-C6	4-M-C8	

… and specifying mixing rule

- Fluid parameters are read from the database using the function
  - createDatabase(reset parameters)
  - Where reset parameter is either 0 if fluid is the same as in previous calculation or 1 if all parameters should be reset. Reading new parameters will take some extra time – but if time is not limiting setting the parameter to 1 can always be used.
- The mixing rule are specified using the method
  - setMixingRule(mixing rule number)
  - Where mixing rule number are one of:
    1. Classic all k_{ij}=0
    2. Classic using k_{ij}
    3. Classic with temperature dependent k_{ij}
    4. Huron Vidal mixing rule using NRTL GE-model (as PVTsim)
    7. CPA - classic with temperature independent k_{ij}
    9. CPA - classic with temperature dependent k_{ij}
    9. CPA - classic with composition and temperature dependent k_{ij}

```
fluid_1.createDatabase(1);
fluid_1.setMixingRule(2);
```

Putting it together..
Select thermodynamic model, create a fluid and add components

```
pressure = 10.0;                                    % pressure in bara
temperature = 273.15;                               % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);     % using the SRK-EoS

fluid_1.addComponent('methane', 1.0);               % adding 1 mole/second of methane
fluid_1.addComponent('propane', 1.0);               % adding 1 mole/second of propane

fluid_1.createDatabase(1);                          % reading new parameters from database
fluid_1.setMixingRule(2);                           % using classic mixing rule with kij
```

Property calculations for single phase systems

- If a fluid is know to have only one phase – with known phase type (gas or liquid), calculations of properties is done using the method:
  - fluidName.init(1/2/3)
  - init(1) – will calculate volume/density and fugasity coefficients
  - init(2) – will calculate additinally to init(1) calculate temperature and pressure derivatives of fugacity coefficients
  - init(3) – will additionally to init(2) calculate compsitional derivatives, such as derivative of fugacity with respect to composition
- On a one phase systems – the number and type of phase has to be set
- If physical properties (viscosity, conductivity, etc.) are needed the methods: fluidName.initPhysicalProperties() has to be called befor retreieving the properties.

```
pressure = 10.0;                                          % pressure in bara
temperature = 273.15;                                     % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);           % using the SRK-EoS

fluid_1.addComponent('methane', 1.0);                     % adding 1 mole/second of methane
fluid_1.addComponent('propane', 1.0);                     % adding 1 mole/second of propane

fluid_1.createDatabase(1);                                % reading new parameters from database
fluid_1.setMixingRule(2);                                 % using classic mixing rule with kij

fluid_1.init(0);                                          % a fluid need to be initialized using init(0)
fluid_1.setNumberOfPhases(1)                              % setting number of phases to one
fluid_1.setPhaseType(0,1)                                 % setting first phase (0) to be of type gas (1)/liquid(0)

fluid_1.init(3);                                          % calculates properties of the fluid
fluid_1.getPhase(0).getEnthalpy();                        % reads the enthalpy

fluid_1.initPhysicalProperties();                         % calculates physical properties
fluid_1.getPhase(0).getPhysicalProperties().getViscosity() % returns the calculated viscosity
```

Reading thermodynamic properties

[can be read after calling init(1/2/3)]

fluid_1.getPhase(0).getEnthalpy(); – enthalpy with unit J

fluid_1.getPhase(0).getComponent('methane').getx(); - get molefraction of methane in phase 0

fluid_1.getPhase('gas').getComponent(0).getx(); - get molefraction of component number 0 (first added component) in gas phase

fluid_1.getPhase('gas').getComponent(0).getz(); - get total feed fraction of component number 0 (first added component)

fluid_1.getPhase(0).getComponent(0).getNumberOfMolesInPhase(); - get number of moles of methane in phase 0

fluid_1.getPhase(0).getComponent(0).getNumberOfMoles(); - get number of moles of component 0 in total fluid

fluid_1.getPhase(0).getZ(); - get compressibility factor of phase 0
fluid_1.getPhase(0).getVolume(); - get volume of phase 0 (unit m^3/1e5)

fluid_1.getVolume(); - get volume of fluid (unit m^3/1e5)

fluid_1.getPhase(0).getComponent(0).getFugicityCoefficient(); - get fugacity coeffisient of component 0 in first phase

fluid_1.getPhase(0).getComponent(0).getdfugdt(); - get derivative of ln fugacity coeffisient of component 0 with respect to temperature

fluid_1.getPhase(0).getComponent(0).getdfugdp - get derivative of ln fugacity coeffisient of component 0 with respect to pressure

fluid_1.getPhase(0).getComponent(0).getdfugdn(0) - get derivative of ln fugacity coeffisient of component 0 with respect to molnumer of component number 1

All available methods for an object can be found by the Matlab function methods(oject) – eg. methods(fluid_1)

Reading physical properties

[can be read after calling initPhysicalProperties()]

fluid_1.getPhase(0).getPhysicalProperties().getViscosity() - *calculated viscosity of phase 0*

fluid_1.getPhase(0).getPhysicalProperties().getDensity() - *calculated density of phase 0*

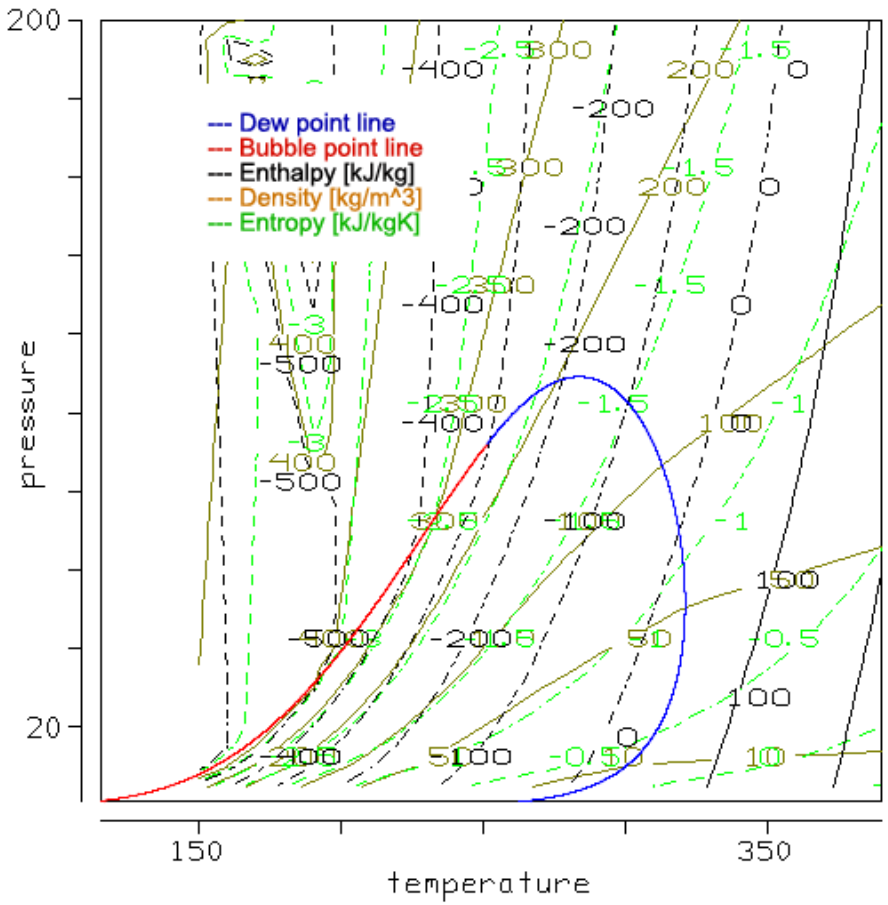
fluid_1.getPhase(0).getPhysicalProperties().getConductivity() - *calculated density of phase 0*

fluid_1.getPhase(0). getInterphaseProperties().getInterfacialTension(0,1) - *calculated interfacial tension between phase 0 and phase 1*

fluid_1.getPhase(0). getInterphaseProperties().getEffectiveDIffusionCoefficient(0) - *calculated effective diffusion coefficient of component 0 in phase 1*

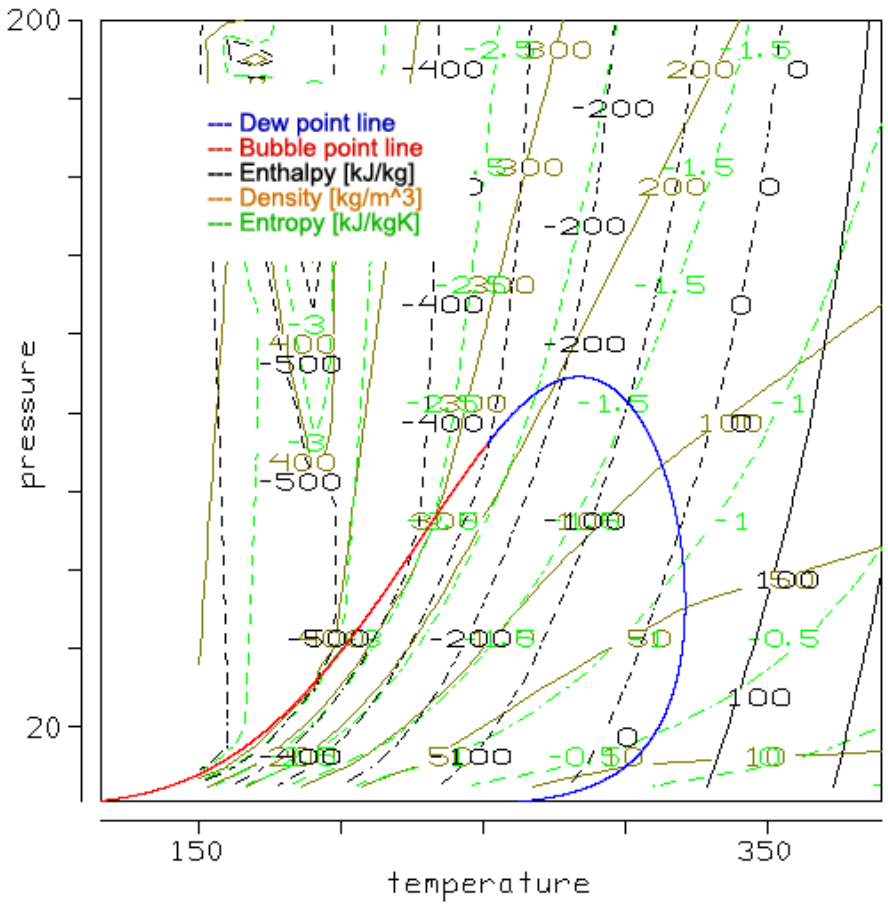
Equilibrium flash calculations

- Flash calculations: TP, PH, PS, TV
- Multiphase flash
- Bubble/dew point, phase envelope



Equilibrium flash calculations

- Flash calculations: TP, PH, PS, TV
- Multiphase flash
- Bubble/dew point, phase envelope



Equilibrium flash caculations

- Flash calculation at constant temperature and pressure is done using the function
  - TPflash(fluid);
- A table with selected fluid properties can be presented graphically by typing the fluid name without a semicolon (see figure)

```
TPflash(fluid_1);
fluid_1
```

System-Report

	Phase 1	Phase 2	Phase 3	Unit
	gas			
methane	5E-1			[-]
propane	5E-1			[-]
Density	1,46758E1			[kg/m^3]
PhaseFraction	1E0			[-]
MolarMass	3,007E1			[kg/kmol]
Cp	1,84944E0			[kJ/kg*K]
Viscosity	8,77119E-6			[kg/m*sec]
Conductivity	2,18291E-2			[W/m*K]
SurfaceTension				[N/m]
Pressure	10.0			[bar]
Temperature	273.15			[K]
Model	SRK-EOS			-
Mixing Rule	classic			-
Stream	DefaultName			-

Flash calculation sepcifications

Specification 1	Specification 2	Method name
Temperature	Pressure	TPflash(fluid name)
Pressure	Enthalpy	PHflash(fluid name, enthalpy)
Pressure	Entropy	PSflash(fluid name, entropy)
Temperature	Volume	TVflash(fluid name, volume)

Example of a combination of a temperature-pressure and a pressure-enthalpy flash:

```
pressure = 10.0;                                    % pressure in bara
temperature = 273.15;                               % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);     % using the SRK-EoS

fluid_1.addComponent('methane', 1.0);               % adding 1 mole/second of methane
fluid_1.addComponent('propane', 1.0);               % adding 1 mole/second of propane

fluid_1.createDatabase(1);                          % reading new parameters from database
fluid_1.setMixingRule(2);                           % using classic mixing rule with kij

TPflash(fluid_1)                                    % doing a flash at constant pressure and temperature

initialEnthalpy = fluid_1.getEnthalpy;              % reading enthalpy of fluid
fluid_1.setPressure(1.0);                           % setting pressure to 1.0 bara
PHflash(fluid_1, initialEnthalpy);                  % doing a flash at given enthalpy and pressure
fluid_1.getTemperature                              % reading resulting temperature
```

Link: TP_PH_flash.m

Multiphase flash calculations

- By default NeqSim will have a maximum of two fluid phases (gas/liquid)
- In many cases in oil industry we will need to calculate equilibrium involving three fluid phases- A typical situation will be when we have a gas, oil and a water phase
- A multiphase calculation is more computational demanding calculations – and the speed of calculations will be reduced
- To turn on the mulitphase calculation option, use the command
    - setMultiPhaseCheck(0/1) - where 0 is off and 1 is on

```
pressure = 10.0;                                        % pressure in bara
temperature = 273.15;                                   % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);         % using the SRK-EoS

fluid_1.addComponent('methane', 1.0);                   % adding 1 mole/second of methane
fluid_1.addComponent('n-heptane', 1.0);                 % adding 1 mole/second of n-heptane
fluid_1.addComponent('water', 1.0);                     % adding 1 mole/second of water

fluid_1.createDatabase(1);                              % reading new parameters from database
fluid_1.setMixingRule(2);                               % using classic mixing rule with kij
fluid_1.setMultiPhaseCheck(1)   % specifies that calculations should check for more than two fluid phases

TPflash(fluid_1)                                        % doing a multi phase flash at constant pressure and
```

Hydrocarbon bubble/dew point, phase envelopes

- The recomended thermodynamic model for calculating hydrocarbon dew points is the UMR-PRU model.
- This model is selected in neqsim using the method:
  thermo('UMR-PRU-EoS',temperature, pressure);

```
pressure = 35.0;                                          % pressure in bara
temperature = 273.15;                                     % temperature in Kelvin

fluid_1 = thermo('UMR-PRU-EoS', temperature , pressure );  % using the UMR-PRU model

fluid_1.addComponent('CO2', 3.0);
fluid_1.addComponent('methane', 90.0);
fluid_1.addComponent('ethane', 5.0);
fluid_1.addComponent('propane', 3.0);

fluid_1.createDatabase(1);                                % reading new parameters from database
fluid_1.setMixingRule('HV', 'UNIFAC_UMRPRU');             % using the mixing rule defined for UMR-PRU¨

dewt(fluid_1);                                            % calculating dew point of fluid
bubt(fluid_1);                                            % calculating bublepoint temperature of fluid
phaseenvelope(fluid_1);                                   % calculating phase envelope of fluid
```

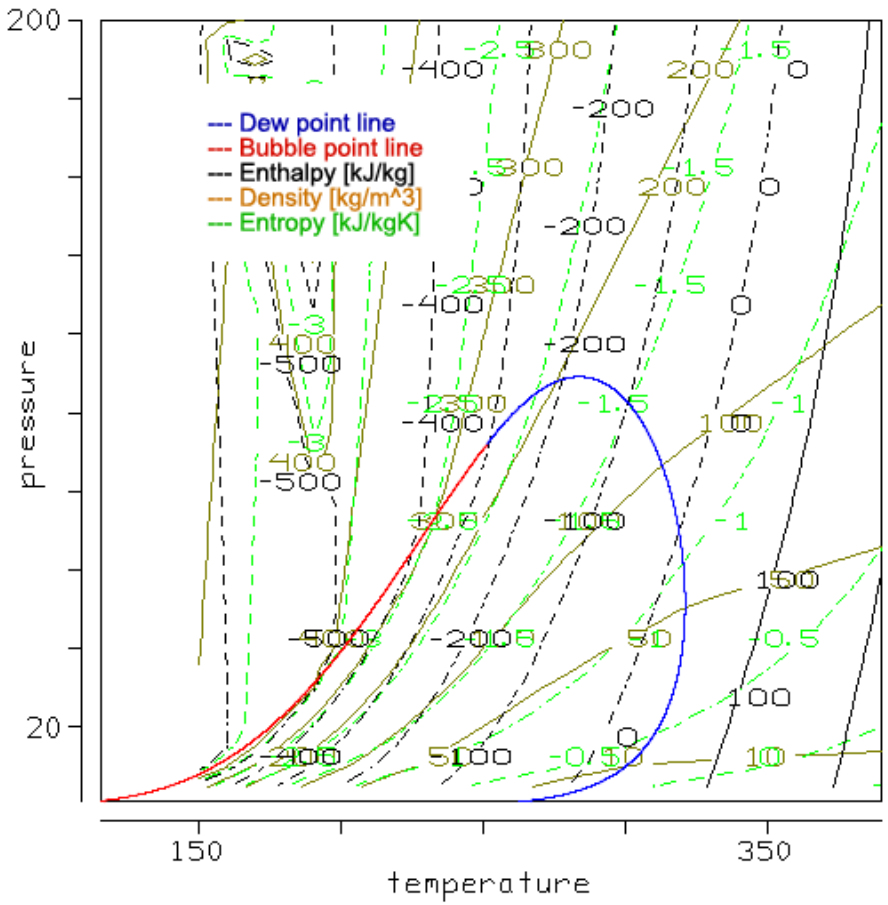
Freezing in hydrocarbon systems (LNG)

- Soid formation temperature can be calculated for CO2, methane, ethane, propane, benzene, toluene, c-hexane, etc.
- Freezing can be a components in low temperatures can be simulatid using the method: freezet(component name)

```
pressure = 35.0;                                          % pressure in bara
temperature = 123.15;                                     % temperature in Kelvin

fluid_1 = thermo('UMR-PRU-EoS', temperature , pressure );  % using the UMR-PRU model

fluid_1.addComponent('CO2', 0.01);
luid_1.addComponent('methane', 100.0);

fluid_1.createDatabase(1);                                % reading new parameters from database

freezet('CO2');                                           % calculating freezing point temperatue of CO2
```

Thermodynamic calculations with water

- Thermodynamic model selection
- Water saturation, water content of a reservoir
- Hydrate equilibrium
- Water, ice and hydrate dew points



Thermodynamic model selection for mixtures containing water

- During the last 15 years advanced and more accurate models have been developed for thermodynamic calulcations involving polar components.
- Statoil has been active in the development of the CPA-EoS (cubic plus assciation)
- The CPA-EoS is the recomended thermodunamic model i Statoil for doing calculations such as:
  - Water content of gas and oil
  - Hydrate equilibrium in complex fluid mixtures (eg. low water content fluids)
  - Distribution of chemicals (such as glycols) in gas and oil
- The CPA-model is selected by specifying the model:
  thermo('cpa', temperature, pressure)
- *By specifying the mixing rule choice as number 9, the classic mixing rule with temperature dependent kij is used*

Water saturation – calculating water content of a reservoir fluid

- Saturating of a gas with water is done using the function *saturateWithWater(fluidName)*
- *The fluid will be saturated with water at current temperature and pressure of the fluid*

```
reservoirPressure = 210.0;                                          % pressure in bara
treservoirTemperature = 373.15;                                     % temperature in Kelvin

fluid_1 = thermo('cpa', treservoirTemperature , reservoirPressure );   % using the CPA-EoS

fluid_1.addComponent('CO2', 3.0);
luid_1.addComponent('methane', 90.0);
fluid_1.addComponent('ethane', 5.0);
fluid_1.addComponent('propane', 3.0);

fluid_1.createDatabase(1);                          % reading new parameters from database
fluid_1.setMixingRule(9);                           % using classic mixing rule with temperature dep. kij

saturateWithWater(fluid_1)                          % saturating the gas with water
```

Water, ice and hydrate dew points

- In the natral gas industry we normally define the water dew point as the first water rich phase that drops out of a gas. This can be aqueous water, ice or gas hydrate.
- The calculation of water dew points are done using the mehods
  - waterDewt(fluidName) – for aqueous dew point
  - hydt(fluidName) – for natural gas hydrate formation
  - Freezt(fluidName, 'water') – for checking ice formation temperature

```
waterDewt(fluid_1);          % calculating aqueous water dew point temperature
hydt(fluid_1);               % calculating the hydrate equilibrim temperature
freezt(fluid_1, 'water');    % calculation of ice formation temperature
```

Thermodynamic calculations with methanol and glycol (TEG, MEG)

- The following glycols and alcohols are implemented in NeqSim
  - Glycols: MEG, TEG, DEG, TREG
  - Alcohols: methanol, ethanol
- Mixtures of water, glycols and alcohols can also be used

Thermodynamic model selection for systems containing glycol and alcohols

- As for calculation involving water – the prefered model for phase equilibrium involving glycols or alcohols suc as methanol, will be the CPA-EoS
- The model sould be able to calculate distributin of water, glycols and alcohols with good accuracy

Phase equilibrium of gas, oil and glycol

- An example of a phase equilibrium calculation of gas, liquid hydrocarbon and MEG and water is illustrated below

```
pressure = 100.0;                                   % pressure in bara
temperature = 293.15;                               % temperature in Kelvin

fluid_1 = thermo('cpa', temperature , pressure );   % using the CPA-EoS

fluid_1.addComponent('CO2', 3.0);
luid_1.addComponent('methane', 90.0);
fluid_1.addComponent('ethane', 5.0);
fluid_1.addComponent('propane', 3.0);

fluid_1.addComponent('n-nonane', 3.0);              % adding heavy hydrocarbon component

fluid_1.addComponent('MEG', 10.0);
fluid_1.addComponent('water', 90.0);

fluid_1.createDatabase(1);          % reading new parameters from database
fluid_1.setMixingRule(9);           % using classic mixing rule with temperature dep. kij
fluid_1.setMultiPhaseCheck(1);      % setting the algorithm to check for more than two phases

TPflash(fluid_1);                   % performing a TP-flash calculation
```

Ice, solid glycol, alcohols and complex freezing points

- Formation of ice, solid glycol and complex solids (combination of water – alcohol/glycol) can be done
- The method used for calculating solid formation of glycols and alcohols is freezt(fluidName, componentName)
- The complex phase is calcuated using the method solidComplexT(fluidName, 'complex component 1', 'complex component 2')
  - The implemented complexes are MEG-water, TEG-water, methanol-water
- Hydrate temeprature is calculated using hydt(fluid)

```
TPflash(fluid_1);                          % performing a TP-flash calculation
freezt(fluidName, 'sold component name')   % calculates solid formation temperature
solidComplexT(fluid_1, 'MEG', 'water')     % cakculates the solid complex formation temperature of MEG-water
```

Calculation of thermodynamic, physical and transport properties

- A numer of physical and transport properties can be calculated using NeqSim. Examples of such properties are:
  - Densities
  - Viscosities
  - Conductivities
  - Diffusivities
  - Interfacial tension
  - Solid surface adsorption properties
- A number of methods are available for calculating the various properties

Density, Enthalpy, Entropy, etc.

- The density of a fluid can be obtained in two ways:
  - fluidName.getPhase(phaseNumber).getPhysicalProperties().getDensity()
  - density(fluidName) – returns an aray of denisties for a mulit phase system
- The enthalp and entropy are obtained from the functions:
  - fluidName.getEnthalpy, fluidName.getEntropy - returns total enthalpy/entropy
  - fluidName.getPhase(phaseNumber).getEnhalpy - returns enthalpy/entropy for a given phase

```
pressure = 10.0;                                        % pressure in bara
temperature = 273.15;                                   % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);         % using the SRK-EoS

fluid_1.addComponent('methane', 1.0);                   % adding 1 mole/second of methane
fluid_1.addComponent('n-heptane', 1.0);                 % adding 1 mole/second of n-heptane
fluid_1.createDatabase(1);                              % reading new parameters from database
fluid_1.setMixingRule(2);                               % using classic mixing rule with kij

TPflash(fluid_1)                                        % doing a TPflash at constant pressure and temperature

fluid_1.getEnthalpy;                                    % reads total Enthalpy (unit is J/mol)
fluid_1.getEntropy;                                     % reads total Entropy
fluid_1.getPhase(0).getEntropy                          % read entropy of first phase (gas in this case)
flluid_1.getPhase(0).getPhysicalProperties().getDensity %reads density of first phase unit is kg/m^3
```

Viscosity, Conductivity

- Viscosity and conductivity is obtained using the functions

  getViscosity(), getConductivity()

Interfacial tension

- Surface tension between phase 0 and 1 is calculated from method:

  fluid_1.getInterphaseProperties().getSurfaceTension(0, 1)

- Various surface tension methods are available (parachor method, gradient theory, etc.). They method to be used are set by:

  fluid_1.getInterphaseProperties().setInterfacialTensionModel(0/1/2);

  where 0 is parachor method, 1 is gradient theory and 2 is linear gradient theory

Solid adsorption

Adsorption of gas on a solid material (kg component/kg material) is calculated by methods:

```
fluid_1.getInterphaseProperties().initAdsorption();
fluid_1.getInterphaseProperties().setSolidAdsorbentMaterial("AC");  % Activated carbon Norit R1
fluid_1.getInterphaseProperties().calcAdsorption();
```

Diffusion coefficients

Characterization, PVT simulation and reservoir fluid tuning

- NeqSim have implemented a number of characterisation methods for the available equation of states
- Typically a fluid consists of standard defined components, pseudo components (oil fractions) and eventually a plus fraction component
- The characterization method will dstiribute components into a number of boiling point fraction (TBP-fractions). As default twelve TBP fractions are used.
- The properties of TBP fractions are calculted using vaius methods relating critical properties (Tc, Pc, acentric factor) to molar mass and density of the fraction.

TBP fractions and model selection

- A pseudocomponent (true boiling point fraction) is added to a fluid using the method:
  addTPBfraction('fraction name', numer of moles, molar mass, density)
  the unit for molar mass is kg/mol and density is gr/cm^3
- The method used for characterization of TBP fractions is set by:
  fluidName.getCharacterization().setTBPModel('PedersenSRK');
  Available options are ,RiaziDaubert, PedersenPR, PedersenSRK.

```
pressure = 10.0;                                          % pressure in bara
temperature = 273.15;                                     % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);           % using the SRK-EoS
fluid_1.getCharacterization().setTBPModel('PedersenSRK');  % setting characterization method (method to calculate Tc, Pc, omega

fluid_1.addComponent('methane', 10.0);                    % adding 1 mole/second of methane
fluid_1.addTBPfraction('C7', 1.0, 0.102, 0.81);           % adding 1 mole/second of a pseudo component C7
fluid_1.addTBPfraction('C8', 0.4, 0.112, 0.83);           % adding 1 mole/second of a pseudo component C8
fluid_1.addTBPfraction('C9', 0.2, 0.132, 0.84);           % adding 1 mole/second of a pseudo component C9

fluid_1.createDatabase(1);                                % reading parameters from database
fluid_1.setMixingRule(2);                                 % using classic mixing rule with kij

TPflash(fluid_1)                                          % doing a TPflash at constant pressure and temperature\
```

Defining and characterization of a fluid with a plus fractions

- A plus fraction component is set by the method
  addPlusFraction('fraction name', numer of moles, molar mass, density)
- *The lumping method is set by the method:*
  fluidName.getCharacterization().setLumpingModel('no');
  available options are: no/ab lumping/pedersen
- *The number of lumped components are set by the method:*
  fluidName.getCharacterization().getLumpingModel().setNumberOfLumpedComponents(number);
- *Characterization of the fluid is done by calling the method:*
  fluidName.getCharacterization().characterisePlusFraction();

```
pressure = 10.0;                                                    % pressure in bara
temperature = 273.15;                                               % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);                     % using the SRK-EoS
fluid_1.getCharacterization().setTBPModel('PedersenSRK');           % setting characterization method (method to calculate Tc, Pc, omega)
fluid_1.getCharacterization().setLumpingModel('pedersen')           % setting lumping model to pedersen
fluid_1.getCharacterization().getLumpingModel().setNumberOfLumpedComponents(8); % set number of lumped components to 8

fluid_1.addComponent('methane', 10.0);                              % adding 1 mole/second of methane
fluid_1.addTBPfraction('C7', 1.0, 0.102, 0.81);                     % adding 1 mole/second of a pseudo component C7
fluid_1.addTBPfraction('C8', 0.4, 0.112, 0.83);                     % adding 1 mole/second of a pseudo component C8
fluid_1.addTBPfraction('C9', 0.2, 0.132, 0.84);                     % adding 1 mole/second of a pseudo component C9
fuid_1.addPlusFraction('C10', 1.0, 0.190, 0.87);                    %adding plus fraction C10+

fluid_1.createDatabase(1);                                          % reading parameters from database
fluid_1.setMixingRule(2);                                           % using classic mixing rule with kij

fluid_1.getCharacterization().characterisePlusFraction();           % characterisation of the fluid
TPflash(fluid_1)                                                    % doing a TPflash at constant pressure and temperature\
```

PVT simulation

- Various types of PVT simulation calculations are implemented in NeqSim:
  - Constand volume depletion
  - Constant mass depletion
  - Saturation pressure simulation
  - Separator test simulation
  - Slim tube simulation

```
pressure = 10.0;                                          % pressure in bara
temperature = 273.15;                                     % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);           % using the SRK-EoS
fluid_1.getCharacterization().setTBPModel('PedersenSRK');  % setting characterization method (method to calculate Tc, Pc, omega)
fluid_1.getCharacterization().setLumpingModel('pedersen')  % setting lumping model to pedersen
fluid_1.getCharacterization().getLumpingModel().setNumberOfLumpedComponents(8); % set number of lumped components to 8

fluid_1.addComponent('methane', 10.0);                    % adding 1 mole/second of methane
fluid_1.addTBPfraction('C7', 1.0, 0.102, 0.81);           % adding 1 mole/second of a pseudo component C7
fluid_1.addTBPfraction('C8', 0.4, 0.112, 0.83);           % adding 1 mole/second of a pseudo component C8
fluid_1.addTBPfraction('C9', 0.2, 0.132, 0.84);           % adding 1 mole/second of a pseudo component C9
fuid_1.addPlusFraction('C10', 1.0, 0.190, 0.87);          %adding plus fraction C10+

fluid_1.createDatabase(1);                                % reading parameters from database
fluid_1.setMixingRule(2);                                 % using classic mixing rule with kij

fluid_1.getCharacterization().characterisePlusFraction();  % characterisation of the fluidConstantVolumeDepletion(tempSystem)

constantVolumeDepletion(fluid_1);                         % Constant volume depletion simulation
constantMassDepletion(fluid_1);                           % constant mass depletion
saturationPressure(fluid_1);                              % saturation pressure
```

PVT fluid tuning

- Tuning of a fluid to PVT data can be done using the NeqSim implemented PVTtuning functions. Example of use of the tuning function are illustrated below for tuning to saturation temperature and pressure (eg. Reservoir conditions)

```
pressure = 10.0;                                          % pressure in bara
temperature = 273.15;                                     % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);           % using the SRK-EoS
fluid_1.getCharacterization().setTBPModel('PedersenSRK');  % setting characterization method (method to calculate Tc, Pc, omega)
fluid_1.getCharacterization().setLumpingModel('pedersen')  % setting lumping model to pedersen
fluid_1.getCharacterization().getLumpingModel().setNumberOfLumpedComponents(8); % set number of lumped components to 8

fluid_1.addComponent('methane', 10.0);                    % adding 1 mole/second of methane
fluid_1.addTBPfraction('C7', 1.0, 0.102, 0.81);           % adding 1 mole/second of a pseudo component C7
fluid_1.addTBPfraction('C8', 0.4, 0.112, 0.83);           % adding 1 mole/second of a pseudo component C8
fluid_1.addTBPfraction('C9', 0.2, 0.132, 0.84);           % adding 1 mole/second of a pseudo component C9
fuid_1.addPlusFraction('C10', 1.0, 0.190, 0.87);          %adding plus fraction C10+

fluid_1.createDatabase(1);                                % reading parameters from database
fluid_1.setMixingRule(2);                                 % using classic mixing rule with kij

fluid_1.getCharacterization().characterisePlusFraction();  % characterisation of the fluidConstantVolumeDepletion(tempSystem)

sattuning = saturationPressure(fluid_1);                  % initiating saturation pressure simulation
sattuning.setSaturationConditions(saturationTemperature, saturationPressure); % setting saturation pressure and temperature
sattuning.run();                                          % running tuning of fluid (default is to tune molecular mass of plus fraction)
```

Hydrate modelling with NeqSim

- NeqSim is well suited for doing hydrate prediction in systems of water ,inhibitors and salts
- The models implemented are based on the CPA-EoS combined with a hydrate model

Hydrate equilibrium calculations

- Hydrate equilibrium temperature calculation for a fluid is done using the function *hydt(fluidName)*
- *The fluid has to be initialized to check for hydrates. This is done using the function setHydrateCheck(0/1) – 0 means do not check for hydrates and 1 means check for hydrates*

```
reservoirPressure = 210.0;                                          % pressure in bara
treservoirTemperature = 373.15;                                     % temperature in Kelvin

fluid_1 = thermo('cpa', treservoirTemperature , reservoirPressure );   % using the CPA-EoS

fluid_1.addComponent('CO2', 3.0);
luid_1.addComponent('methane', 90.0);
fluid_1.addComponent('ethane', 5.0);
fluid_1.addComponent('propane', 3.0);

fluid_1.createDatabase(1);          % reading new parameters from database
fluid_1.setMixingRule(9);           % using classic mixing rule with temperature dep. kij

saturateWithWater(fluid_1);         % saturating the gas with water

fluid_1.setHydrateCheck(1);         % initializing the fluid to check for hydrates
fluid_1.setMultiPhaseCheck(1);      % setting the algorithm to check for more than two phases

hydt(fluid_1);                      % calculating the hydrate equilibrim temperature
```

Hydrate equilibrium calculations with inhibitors

equinor

- NeqSim is well suited for performing hydrate calulcations and evaluating the effect of adding inhibitors like glycols or alcohols
- Hydrate equilibrium temperature is calculated using hydt(fluid). Hydrate equilibrium pressure is calculated using hydp(fluidName)
- The calculated structure can be obtained usin the function: *fluid.getPhase('hydrate').getHydrateStructure()*

```
pressure = 100.0;                                        % pressure in bara
temperature = 293.15;                                    % temperature in Kelvin

fluid_1 = thermo('cpa', temperature , pressure );        % using the CPA-EoS

fluid_1.addComponent('CO2', 3.0);
luid_1.addComponent('methane', 90.0);
fluid_1.addComponent('ethane', 5.0);
fluid_1.addComponent('propane', 3.0);

fluid_1.addComponent('n-nonane', 3.0);                   % adding heavy hydrocarbon component

fluid_1.addComponent('MEG', 10.0);
fluid_1.addComponent('water', 90.0);

fluid_1.createDatabase(1);                   % reading new parameters from database
fluid_1.setMixingRule(9);                    % using classic mixing rule with temperature dep. kij
fluid_1.setMultiPhaseCheck(1);               % setting the algorithm to check for more than two phases
fluid_1.setHydrateCheck(1);                  % initializing the fluid to check for hydrates

hydt(fluid_1);                               % performing a TP-flash calculation
fluid_1.getPhase('hydrate').getHydrateStructure  % read hydrate structure
```

Hydrate equilibrium calculations with salts

- Salts are added to the system using the function:
  addSalt('NaCl', moles);

```
pressure = 100.0;                                        % pressure in bara
temperature = 293.15;                                    % temperature in Kelvin

fluid_1 = thermo('cpa', temperature , pressure );        % using the CPA-EoS

fluid_1.addComponent('CO2', 3.0);
luid_1.addComponent('methane', 90.0);
fluid_1.addComponent('ethane', 5.0);
fluid_1.addComponent('propane', 3.0);

fluid_1.addComponent('n-nonane', 3.0);                   % adding heavy hydrocarbon component

fluid_1.addComponent('MEG', 10.0);
fluid_1.addComponent('water', 90.0);

fluid_1.addSalt('NaCl', 0.1);                            % adding salt 0.1 mole/sec

fluid_1.createDatabase(1);                   % reading new parameters from database
fluid_1.setMixingRule(9);                    % using classic mixing rule with temperature dep. kij
fluid_1.setMultiPhaseCheck(1);               % setting the algorithm to check for more than two phases
fluid_1.setHydrateCheck(1);                  % initializing the fluid to check for hydrates

hydt(fluid_1);                               % performing a TP-flash calculation
fluid_1.getPhase('hydrate').getHydrateStructure  % read hydrate structure
```

Top of line hydrate equilibrium

- *Top of line hydrate equilibrium temperature is calculated using function: hydt_TOL(fluidname)*

```
pressure = 100.0;                                   % pressure in bara
temperature = 293.15;                               % temperature in Kelvin

fluid_1 = thermo('cpa', temperature , pressure );   % using the CPA-EoS

fluid_1.addComponent('CO2', 3.0);
luid_1.addComponent('methane', 90.0);
fluid_1.addComponent('ethane', 5.0);
fluid_1.addComponent('propane', 3.0);

fluid_1.addComponent('n-nonane', 3.0);              % adding heavy hydrocarbon component

fluid_1.addComponent('MEG', 10.0);
fluid_1.addComponent('water', 90.0);

fluid_1.createDatabase(1);          % reading new parameters from database
fluid_1.setMixingRule(9);           % using classic mixing rule with temperature dep. kij
fluid_1.setMultiPhaseCheck(1);      % setting the algorithm to check for more than two phases
fluid_1.setHydrateCheck(1);         % initializing the fluid to check for hydrates

hydt_TOL(fluid_1);                  % calculates top of line hydrate formation temperature
```

Wax and asphaltene calculations

- The simlified PC-SAFT equation of stat is recomended for doing wax calculations in NeqSim. The method is beleived to be accurate for long chained parafinic hydrocarbons. The simlified PC-SAFT method is selectd by spesifying:

```
pressure = 10.0;            % pressure in bara
temperature = 273.15;       % temperature in Kelvin

fluid_1 = thermo('sPC-SAFT', temperature, pressure);
```

- Asphaltene calculations can be done be selecting the SRK-EoS (no special recomended model for asphaltene precipitation at the moment)

Wax calculations

- Thermodynamics of wax are simulated following the methods of Pedersen et. al.
- Wax calculations need to be initiated using the method:
  fluidName.addTBPWax(); - to split the fractions up in wax formers and non-wax
  fluidName.addSolidComplexPhase("wax");
- Wax equilibrium temperature is calculated using the method:
  waxt(fluidName);

```
pressure = 10.0;                                          % pressure in bara
temperature = 273.15;                                     % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);           % using the SRK-EoS
fluid_1.getCharacterization().setTBPModel('PedersenSRK');  % setting characterization method (method to calculate Tc, Pc, omega)
fluid_1.getCharacterization().setLumpingModel('pedersen')  % setting lumping model to pedersen
fluid_1.getCharacterization().getLumpingModel().setNumberOfLumpedComponents(8); % set number of lumped components to 8

fluid_1.addComponent('methane', 10.0);                    % adding 1 mole/second of methane
fluid_1.addTBPfraction('C7', 1.0, 0.102, 0.81);           % adding 1 mole/second of a pseudo component C7
fluid_1.addTBPfraction('C8', 0.4, 0.112, 0.83);           % adding 1 mole/second of a pseudo component C8
fluid_1.addTBPfraction('C9', 0.2, 0.132, 0.84);           % adding 1 mole/second of a pseudo component C9
fuid_1.addPlusFraction('C10', 1.0, 0.190, 0.87);          %adding plus fraction C10+

fluid_1.createDatabase(1);                                % reading parameters from database
fluid_1.getCharacterization().characterisePlusFraction();  % characterisation of the fluidConstantVolumeDepletion(tempSystem)
fluid_1.addTBPWax();                                      % Splitting up TBPfraction in to wax formers and non wax formers
fluid_1.setMixingRule(2);                                 % using classic mixing rule with kij
fluid_1.addSolidComplexPhase('wax');                      % adding the possibility to simulating waxphases

waxt(fluid_1);                                            % calculating wax equilibrium formation temperature
```

Prediction of asphaltene content

- To be implemented.....

Non-equilibrium calculations

- Non-equilibrium calculations are calculted using the non-equilibrium multipase model developed for NeqSim
- The same base model is used for all non-equilibrium simulations
- Examples of such calculations are evaporation and condensation processes

Simulating two-phase non equilibrium processes

- A fluid is reated adding components to invidual phases using fluid.addComponent(componentName, moles/Sec, phase)
- A pipe is created using pipe(innerDiameter, length)
- Evaporation is modeled using the function neqsim(fluid, pipe, flowtype) - flow type can be selected to auto

```
pressure = 10.0;                                          % pressure in bara
temperature = 273.15;                                     % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);           % using the SRK-EoS
fluid_1.getCharacterization().setTBPModel('PedersenSRK');  % setting characterization method (method to calculate Tc, Pc, omega

fluid_1.addComponent('methane', 10.0, 0);                 % adding 1 mole/second of methane to phase 0 (gas)
fluid_1.addTBPfraction('C7', 1.0, 0.102, 0.81, 1);        % adding 1 mole/second of a pseudo component C7 to phase 1 (liquid)
fluid_1.addTBPfraction('C8', 0.4, 0.112, 0.83, 1);        % adding 1 mole/second of a pseudo component C8 to phase 1 (liquid)
fluid_1.addTBPfraction('C9', 0.2, 0.132, 0.84, 1);        % adding 1 mole/second of a pseudo component C9 to phase 1 (liquid)

fluid_1.createDatabase(1);                                % reading parameters from database
fluid_1.setMixingRule(2);                                 % using classic mixing rule with kij

pipe = pipe(1.0, 100.0);                                  % creating a pipe with inner diameter 1.0 and length 100 meter

sim = neqsim(fluid_1, pipe, 'droplet');                   % simulates a non equilibrium process of the fluid In the pipe
sim.run                                                   % running calculation
sim.display                                               % displaying results
```

Simulating total liquid evaporation processes

- A total evoparation process means that the fluid will be one phase at equilibrium
- A typical example can be evaporation of water into a dry gas

```
pressure = 10.0;                                          % pressure in bara
temperature = 273.15;                                     % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);           % using the SRK-EoS
fluid_1.getCharacterization().setTBPModel('PedersenSRK');  % setting characterization method (method to calculate Tc, Pc, omega

fluid_1.addComponent('methane', 10.0, 0);                 % adding 1 mole/second of methane to phase 0 (gas)
fluid_1.addTBPfraction('C7', 0.01, 0.102, 0.81, 1);       % adding 1 mole/second of a pseudo component C7 to phase 1 (liquid))

fluid_1.createDatabase(1);                                % reading parameters from database
fluid_1.setMixingRule(2);                                 % using classic mixing rule with kij

pipe = pipe(1.0, 100.0);                                  % creating a pipe with inner diameter 1.0 and length 100 meter

sim = neqsim(fluid_1, pipe, 'droplet');                   % simulates a non equilibrium process of the fluid In the pipe
sim.run                                                   % running calculation
sim.display                                               % displaying results
```

Simulating nucleation and droplet growth

- Nucleation is the process where a new phase are created. It can be homoenious (in fluid) or hetrogeneous (in surface) nucleation
- The simulation has to be initiated using flow type ´droplet nucleation´

```
pressure = 10.0;                                          % pressure in bara
temperature = 273.15;                                     % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);           % using the SRK-EoS
fluid_1.getCharacterization().setTBPModel('PedersenSRK');  % setting characterization method (method to calculate Tc, Pc, omega

fluid_1.addComponent('methane', 10.0, 0);                 % adding 1 mole/second of methane to phase 0 (gas)
fluid_1.addTBPfraction('C7', 0.01, 0.102, 0.81, 0);       % adding 1 mole/second of a pseudo component C7 to phase 1 (liquid))
fluid_1.addTBPfraction('C7', 0.01, 0.102, 0.81, 0);       % adding 1 mole/second of a pseudo component C7 to phase 1 (liquid))

fluid_1.createDatabase(1);                                % reading parameters from database
fluid_1.setMixingRule(2);                                 % using classic mixing rule with kij

pipe = pipe(1.0, 100.0);                                  % creating a pipe with inner diameter 1.0 and length 100 meter

sim = neqsim(fluid_1, pipe, ''droplet nucleation');       % simulates a non equilibrium process of the fluid In the pipe
sim.run                                                   % running calculation
sim.display                                               % displaying results
```

Calculation of gas quality properties

- Gas quality parameters such as heating values (GCV), Wobbe index (WI) and relative densities are calculated accurding to ISO6976-2005
- Other gas quality parameters such as dew points (water, hydrocarbons) can also be calculated

TBP fractions and model selection

- A pseudocomponent (true boiling point fraction) is added to a fluid using the method:
  addTPBfraction('fraction name', numer of moles, molar mass, density)
  the unit for molar mass is kg/mol and density is gr/cm^3
- The method used for characterization of TBP fractions is set by:
  fluidName.getCharacterization().setTBPModel('PedersenSRK');
  Available options are ,RiaziDaubert, PedersenPR, PedersenSRK.

```
pressure = 10.0;                                          % pressure in bara
temperature = 273.15;                                     % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);           % using the SRK-EoS
fluid_1.getCharacterization().setTBPModel('PedersenSRK');  % setting characterization method (method to calculate Tc, Pc, omega

fluid_1.addComponent('methane', 10.0);                    % adding 1 mole/second of methane
fluid_1.addTBPfraction('C7', 1.0, 0.102, 0.81);           % adding 1 mole/second of a pseudo component C7
fluid_1.addTBPfraction('C8', 0.4, 0.112, 0.83);           % adding 1 mole/second of a pseudo component C8
fluid_1.addTBPfraction('C9', 0.2, 0.132, 0.84);           % adding 1 mole/second of a pseudo component C9

fluid_1.createDatabase(1);                                % reading parameters from database
fluid_1.setMixingRule(2);                                 % using classic mixing rule with kij

TPflash(fluid_1)                                          % doing a TPflash at constant pressure and temperature\
```

GCV, WI and relative density calculations (ISO6976-2005)

- GCV are calculated using the method
  GCV(fluidName, volume reference temperature, combustion reference temperature)
- WI are calculated using the method
  WI(fluidName, volume reference temperature, combustion reference temperature)
- Relative density are caluated using:
  reldens(fluidName, volume reference temperature)

```
pressure = 35.0;                                        % pressure in bara
temperature = 273.15;                                   % temperature in Kelvin

fluid_1 = thermo('srk', temperature , pressure );   % using the UMR-PRU model

fluid_1.addComponent('CO2', 3.0);
luid_1.addComponent('methane', 90.0);
fluid_1.addComponent('ethane', 5.0);
fluid_1.addComponent('propane', 3.0);

fluid_1.createDatabase(1);                              % reading new parameters from database
fluid_1.setMixingRule(2);                               % using the mixing rule defined for UMR-PRU¨

GCV(fluid_1, 15, 15);                                   % calculating gross calorific value 15C/15C
WI(fluid_1, 15, 15);                                    % calculating wobbe index 15C/15C
reldens(fluid_1, 15);                                   % calculating relative density at 15C
```

Calculating water dew point (ISO 5198)

- GERG-water EOS can be used to calculate water dew point according to ISO5198
- The water dew point is calculated using the method:
  dewt_gw(fluidName)

```
pressure = 290.0;                                   % pressure in bara
temperature = 283.15;                               % temperature in Kelvin

fluid_1 = thermo('srk', temperature , pressure );   % using the CPA-EoS

fluid_1.addComponent('CO2', 3.0);
luid_1.addComponent('methane', 90.0);
fluid_1.addComponent('ethane', 5.0);
fluid_1.addComponent('propane', 3.0);
fluid_1.addComponent('water', 30.0e-4);             % adding 40 ppm water

fluid_1.createDatabase(1);                          % reading new parameters from database
fluid_1.setMixingRule(2);                           % using classic mixing rule with temperature dep. kij

dew_gw(fluid_1)                                     % calculation water dew point with GERG-water model
```

Process simulation in NeqSim

- Process simulation of simple process plants can be dne in NeqSim.
- Various unit operations such as:
  - Straems
  - Separators
  - Compressors
  - Valves
  - Mixers, and more
- Simple dynamic process simulations can be performed
- The solution algorithms solves one unit operation at a time in a predfined sequence
- After defining and connecting the unit operations – process simulation are run using the run() method

Defining streams

- Stream are connecting unit operations. A stream is defined from a fluid using the function: stream(fluidName);
- Two typies of streams are defined: the equilibrium stream (at equilibrium at given T & P), and non-equilibrium stream difned by neqstream(fluidName) where the phases are no assumed to be in equilibrium

```
pressure = 10.0;                                    % pressure in bara
temperature = 273.15;                               % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);     % using the SRK-EoS

fluid_1.addComponent('methane', 1.0);               % adding 1 mole/second of methane
fluid_1.addComponent('propane', 1.0);               % adding 1 mole/second of propane

fluid_1.createDatabase(1);                          % reading new parameters from database
fluid_1.setMixingRule(2);                           % using classic mixing rule with kij

stream_1 = strream(fluid_1);                        % defining a stream

run();                                              % running the prossess simulation (flash stream)
stream_1.displayRsults();                           % displaying reusults of stream
stream_1.getThermoSystem().getPhase(0).getZ()       % read compressibility of gas phase of the stream
```

Valves

- A valve is made using the function:
  valve(inputStream, pressure out) where pressure out is in bara
- The outlet pressure can also be set by the function setOutletPressure(pressure)
- Cv, valve opening, etc can also be specified for the valve
- The outlet stream from a valve is obtained using the function
  valve.getOutStream()

```
pressure = 10.0;                                  % pressure in bara
temperature = 273.15;                             % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);   % using the SRK-EoS

fluid_1.addComponent('methane', 1.0);             % adding 1 mole/second of methane
fluid_1.addComponent('propane', 1.0);             % adding 1 mole/second of propane

fluid_1.createDatabase(1);                        % reading new parameters from database
fluid_1.setMixingRule(2);                         % using classic mixing rule with kij

stream_1 = strream(fluid_1);                      % defining a stream
valve_1 = valve(stream_1, 5.0);                   % setting up a valve and setting 5 bar outlet pres

run();                                            % running the prossess simulation (flash stream)
valve_1.displayResult()                           % displaying results from valve simulation
```

Separators and scrubbers

- Separators are defined using the function separator(streamName)
- A gas scrubber is defined using the function gasscrubber(streamName)
- The separation efficiency can be spesified using the function: setEfficiency()
- Dimension and internals of spearators and scrubbers can also be spesified

```
pressure = 10.0;                                    % pressure in bara
temperature = 273.15;                               % temperature in Kelvin

fluid_1 = thermo('srk', temperature, pressure);     % using the SRK-EoS

fluid_1.addComponent('methane', 1.0);               % adding 1 mole/second of methane
fluid_1.addComponent(n-heptane', 1.0);              % adding 1 mole/second of propane

fluid_1.createDatabase(1);                          % reading new parameters from database
fluid_1.setMixingRule(2);                           % using classic mixing rule with kij

stream_1 = stream(fluid_1);                         % defining a stream
valve_1 = valve(stream_1, 5.0);                     % setting up a valve and setting 5 bar outlet pres
separator_1 = separator(valve_1.getOutStream())     % defining a separator

run();                                              % running the prossess simulation (flash stream)
separator_1.getGasOutStream().displayResults()      % displaying the gas from the separator
```

Compressors and pumps

- A compressor is defined using the function: compressor(streamName, outPressure)
- A pump is made using the function pump(streamName)
- The outlet pressure is set by setOutPressure(pressure)
- The efficiency is set by the function: setEfficiency(efficiency)
- The calculation type can be set to adiabatic, polytropic.

```
stream_1 = stream(fluid_1);                                  % defining a stream
valve_1 = valve(stream_1, 5.0);                              % setting up a valve and setting 5 bar outlet pres
separator_1 = separator(valve_1.getOutStream())              % defining a separator

gasCompressor_1 = compressor(separator_1.getGasOutStream(), 10.0); % setting up a gas compressor
oilPump_1 = pump(separator.getOilOutStream());                     % defining a pump

run();                                                       % running the prossess simulation (flash stream)
```

Heat exchangers, heater and coolers

- A heater is defined using the function heater(streamName, outTemperature)
- The duty can be read usng the function getDuty() -the reported duty will be in Watt
- Alternatively the duty can be specified – and the outlet temperature calculated
- A cooler is made in the same way as a heater
- A heatexchanger can be defined using the function heatexchanger(stream 1, stream 2), and specifying U*A values

```
stream_1 = stream(fluid_1);                                  % defining a stream
valve_1 = valve(stream_1, 5.0);                              % setting up a valve and setting 5 bar outlet pres
separator_1 = separator(valve_1.getOutStream())              % defining a separator

gasCompressor_1 = compressor(separator_1.getGasOutStream(), 10.0); % setting up a gas compressor
oilPump_1 = pump(separator.getOilOutStream());                     % defining a pump

gasHeater_1 = heater(gasCompressor_1.getOutStream(), 100.0);       % setting up a heater

run();                                                       % running the process simulation (flash stream)

gasHeater_1.getDuty();                                       % reading duty of heater
```

Mixers

- A mixer is defined using the function mixer()
- Streams are added to the mixer using the function addStream(streamName);
- Any number of streams can be added to the mixer

```
stream_1 = stream(fluid_1);                                  % defining a stream
valve_1 = valve(stream_1, 5.0);                              % setting up a valve and setting 5 bar outlet pres
separator_1 = separator(valve_1.getOutStream())              % defining a separator

gasCompressor_1 = compressor(separator_1.getGasOutStream(), 10.0); % setting up a gas compressor
oilPump_1 = pump(separator.getOilOutStream());                     % defining a pump

gasHeater_1 = heater(gasCompressor_1.getOutStream(), 100.0);       % setting up a heater

mixer_1 = mixer();                                           % creating a mixer
mixer_1.addStream(gasHeater_1.getOutStream());               % adding the gas stream

run();                                                       % running the process simulation (flash stream)
```

Resirculation streams

- Resirculations stream are added in the same way as normal stream
- The run() method is repeated until convergence is obtained (sucessive substitutuion and sequential solving)