On the Principles of Thermodynamic Modelling

T. Haug-Warberg

Norwegian University of Science and Technology, Department of Chemical Engineering, Trondheim, Norway

INTRODUCTION

The implementation of thermodynamic models is a tedious task. The models are quite complex (multicomponent mixtures) and the data base issue may also cause some confusion (i.e. origin of data, scientific units, standard states, etc.) Whereas the current trend is to develop, and promote, advanced program interfaces (ASPEN, FACT, etc.), a viable alternative could be to export dedicated software. Rather than implementing an *N*-component model which is restricted runtime to, say, a 3-component system, we can export the 3-component model with parameters and physical data into a stand-alone computer program (MATLAB). This strategy offers maximum flexibility to the programmer, provided of course that a set of construction rules exist which ensure feasibility of the exported model.

THERMODYNAMIC ALGEBRA

The proposed construction rules are based on the observation that all(?) thermodynamic models can be derived from the algebraic expression:

$$f(x) = A(x) + B(x) * C(x)^n$$

The symbols *A*, *B* and *C* stand for thermodynamic objects of the type: **Helmholtz**, **EquationOfState**, **ModTVN** (ideal gas, SRK, etc.), **StandardState**, **MuT_cp** (heat capacity integral), **MuT_hg** (enthalpy and Gibbs energy of formation), **MuT_hs** (enthalpy of formation and entropy), etc. The exponent *n* specifies a function modifier. An underlying class inheritance scheme guarantees that the objects are combined in a thermodynamically feasible manner (semantic check). In addition, the following operator conventions must also be obeyed (syntax check):

$$\begin{aligned} A+B &= B+A \\ A+(B_1+B_2) &= A+B_1+B_2 \\ B*C &\neq C*B \\ B*(C*D) &= B*C*D \\ B*(C_1+C_2) &= B*C_1+B*C_2 \end{aligned}$$

OPERATOR OVERLOADING

The construction rules are implemented using a technique called operator overloading. Most modern programming languages (C++, Phyton, Ruby, etc.) has this feature, which makes it possible to define the action *A***B* on any pair of algebraic objects *A* and *B*. Thus, the syntax of the expression is under full control of the programming language (nice because we do not have to invent a new grammar), while the semantics is under control of the programmer (who can concentrate on the operator properties). This combines the best of two worlds and makes it possible to write a very clean interface to the function object *f*(*x*).

EXPORTED CODE

The object *f*(*x*) is stored in an onion-like structure where each shell defines a thermodynamic contribution which holds its own component list, a database and the function code. The function code is stored in a standard data tree with operators sitting on the branch nodes and vectors, matrices, etc. on the leaf nodes. From this standard representation it is straightforward to export the object into XML, LaTeX, Matlab, etc.

1: THERMODYNAMIC MODEL

Presume we need the Helmholtz energy and its derivatives for the system NO-N_2O-NO_2. Using the SRK equation of state, the thermodynamic description would look like:

$$\begin{aligned} A &= A^\circ + A^{ig} + A^{SRK} \\ A^\circ &= \sum_i N_i m_i^\circ \end{aligned}$$

This description is quite adequate at the highest information level, but once the model is going to be implemented there are several questions to be answered:

- Where are the physical data taken from ?
- What is the standard state ?
- Which heat capacity function is used ?
- Which modification of SRK is used ?

2: CANONICAL REPRESENTATION (RUBY CODE)

```
n  = ['nitric-oxide','nitrous-oxide','nitrogen-dioxide']
n1 = n[0..0]
n2 = n[1..-1]
A  = Surface(n) * (
       Helmholtz(n) * [
         StandardState(n) * [
           MuT_cp(n1,:poly3,'ig',:reid77) * (
             MuT_hg(n1,:h0,'ig',:reid87) +
             MuT_hg(n1,:g0,'ig',:dippr96)
           ),
           MuT_cp(n2,:dippr,'ig',:dippr96) * (
             MuT_hs(n2,:h0,'ig',:reid87) +
             MuT_hs(n2,:s0,'ig',:dippr96)
           )
         ],
         EquationOfState(n) * (
           ModTVN_ideal(n,:idealgas,'gas')
         ),
         EquationOfState(n) * (
           ModTVN(n,:srk,'gas',:reid77) **
                  (:m_gd,['gas','a','mfac'],:reid87)
         )
       ]
     )
puts A.to_matlab
puts A.to_latex
A.from_xml(A.to_xml).to_xml == A.xml     # Returns true !!
```

Function · Name tag · "chain" · "patch" · Database · Property · "tell" · SRK modification

3a: EXPORT HIGH LEVEL MATLAB CODE

```
function [S] = ModTVN_ideal_idealgas_58097724(x)

pcirc = [101325.0;101325.0;101325.0];
R     = 8.314511984;
T     = x(1);
V     = x(2);
i     = [3,4,5];
n     = x(i);
NR    = sum(n)*R;
NRT   = NR*T;
e     = [1;1;1];
p     = R*log(R*T*(n./pcirc)/V);
g_1   = n'*p;
g_2   = -NRT/V;
g_i   = T*p;
H_11  = NR/T;
H_21  = -NR/V;
H_i1  = p + R*e;
H_22  = NRT/V^2;
H_i2  = -R*(T/V)*e;
H_ii  = R*T*diag(e./n);
S.g   = [g_1;g_2;g_i];
S.H   = [H_11,H_21',H_i1';H_21,H_22,H_i2'; …
         H_i1,H_i2,H_ii];
```

Reversible export and import of XML code (not shown)

Gradient and Hessian of Helmholtz energy

3b: EXPORT LaTeX DOCUMENTATION

ModTVN-ideal-idealgas-58097724 : $\boldsymbol{x} \curvearrowright S$

$$\boldsymbol{p}^\circ = \begin{pmatrix} \frac{101325.0\ \mathrm{kg}}{\mathrm{m\,s^2}} \\ \frac{101325.0\ \mathrm{kg}}{\mathrm{m\,s^2}} \\ \frac{101325.0\ \mathrm{kg}}{\mathrm{m\,s^2}} \end{pmatrix}$$

$$R = \frac{8.314511984\ \mathrm{kg\,m^2}}{\mathrm{K\,mol\,s^2}}$$

$$T = \boldsymbol{x}_{[1]}$$

$$V = \boldsymbol{x}_{[2]}$$

$$\boldsymbol{i} = \begin{pmatrix} 3 & 4 & 5 \end{pmatrix}$$

$$\boldsymbol{n} = \boldsymbol{x}_{[\boldsymbol{i}]}$$

$$NR = \sum_i (\boldsymbol{n})_{[i]}\, R$$

$$NRT = NRT$$

$$\boldsymbol{e} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

$$\boldsymbol{p} = R \ln\left(\frac{RT\,(\boldsymbol{n} \div \boldsymbol{p}^\circ)}{V}\right)$$

$$g_1 = \boldsymbol{n}^{\mathrm{T}} \boldsymbol{p}$$

$$g_2 = \frac{-NRT}{V}$$

$$g_i = T\boldsymbol{p}$$

$$H_{1,1} = \frac{NR}{T}$$

$$H_{2,1} = \frac{-NR}{V}$$

$$H_{i,1} = \boldsymbol{p} + R\boldsymbol{e}$$

$$H_{2,2} = \frac{NRT}{V^2}$$

$$H_{i,2} = -R\left(\frac{T}{V}\right)\boldsymbol{e}$$

$$H_{i,i} = RT\,(\boldsymbol{e} \div \boldsymbol{n})^{\mathrm{D}}$$

$$S \times g = \begin{pmatrix} g_1 \\ g_2 \\ g_i \end{pmatrix}$$

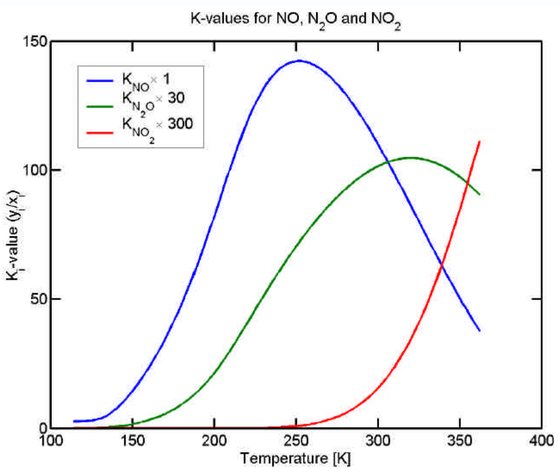
$$S \times H = \begin{pmatrix} H_{1,1} & H_{2,1} & H_{i,1}^{\mathrm{T}} \\ H_{2,1} & H_{2,2} & H_{i,2}^{\mathrm{T}} \\ H_{i,1} & H_{i,2} & H_{i,i} \end{pmatrix}$$

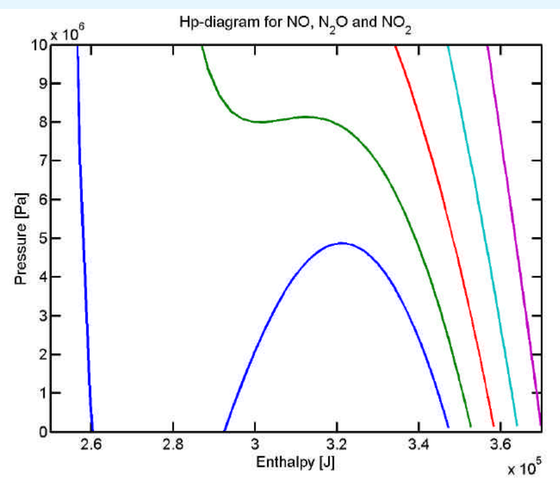
4a: VLE FLASH CALCULATION

```
%Iterate on (negative) pressure and chemical potentials.
while norm(dv./(xv+xl))>1e-7
  V  = Surface(xv);
  L  = Surface(xl);
  dv = [0;inv(L.H(i,i)+V.H(i,i))*(L.g(i)-V.g(i))];
  s  = min(1,-0.8/min([dv./xv;-dv./xl]));
  xv = xv + s*dv;
  xl = xl - s*dv;
end
```

4b: ENTHALPY-PRESSURE CALCULATION

```
h = -t*A.g(1) + A.g(n)'*x(n);
p = -A.g(2);
```





CONCLUSIONS

A fully consistent Helmholtz object has been defined in terms of seven primitive model classes and three algebraic operators (* and + and ^). It has been shown how the object can be exported to XML, LaTeX and Matlab. All the output formats encapsulate the functionality and the parameters needed to calculate the gradient vector and the Hessian matrix of the Helmholtz energy for the given system.

DISCUSSION

In the infancy of this project it was attempted to develop the code in C++. This attempt failed for two reasons: 1) C++ is a huge language which gives the programmer access to both imperative, functional and object oriented programming within one language formalism. This makes it very hard to settle for a concise programming style. 2) C++ is a compiled language with rigid class structures which makes the code highly sensitive to changes in the class design and the basic data types. This proved to be a big drawback in the early stages of the development (when the design changed every week).

Ruby [4] was finally chosen for its compact class abstraction and very clean syntax. This choice has had a fair degree of success, but the models should definitely been implemented in a pure functional language like Haskell. However, there seems to be little activity in functional programming and it is not clear how this issue should be resolved.

FUTURE WORK

In this example the thermodynamic contributions were assembled into a Helmholtz energy surface. The proposed methodology can be extended to Legendre and Massieu transformations [2,3], and thus make a completely general thermodynamic surface (arbitrary coordinates). Finally, the surfaces can be combined into an equilibrium manifold including phase and reaction equilibrium calculations. In all cases gradient and Hessian information will be available. Automatic differentiation of the calculated variables with respect to model parameters is also possible. This is essential for parameter

ACKNOWLEDGEMENTS

Thanks to M.Sc. student Bjørn Tore Løvfall at the Department of Chemical Engineering at NTNU for programming the export facilities, and to the Modelling and Simulation group at the Corporate Reseacrh Centre, Norsk Hydro Porsgrunn, Norway, for continuous support and interest in this project.

REFERENCES

[1] Iverson, K.E., Notation as a Tool of Thought, *Commun. ACM*, **23** (8), 444-465 (1980)
[2] Callen, H., *Thermodynamics and an Introduction to Thermostatistics*, 2nd ed., John Wiley , New York (1985).
[3] Beegle, B.L., Legendre Transforms and Their Applications in Thermodynamics, *AIChE J.*, **20** (6), 1194-1200 (1974).
[4] Matsumoto, Y., *Ruby in a Nutshell*, O'Reilly, (2001).