



NTNU

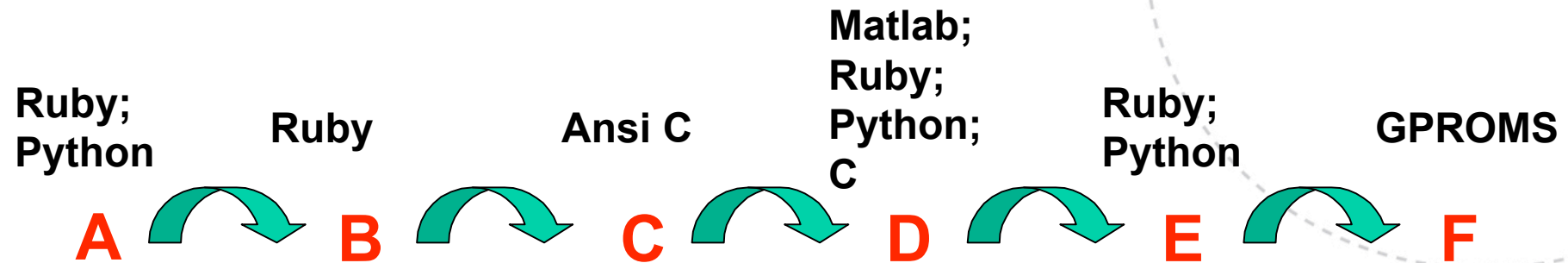
Innovation and Creativity

Textual Formats in Thermodynamic Modelling

**Tore Haug-Warberg
Bjørn Tore Løvfall**

**Department of Chemical Engineering
Spring semester 2008**

The A-F of Process Modelling



Algebraic description of the model parts (flow, thermodynamics, transport, kinetics) at the same abstraction level as is used in ordinary mathematical writings and natural science textbooks

Automatic exportation of the model in the RGrad language (Ruby Gradients) using **B**roadcasting techniques for the symbolic calculation and manipulation of gradients, Hessians and Jacobians

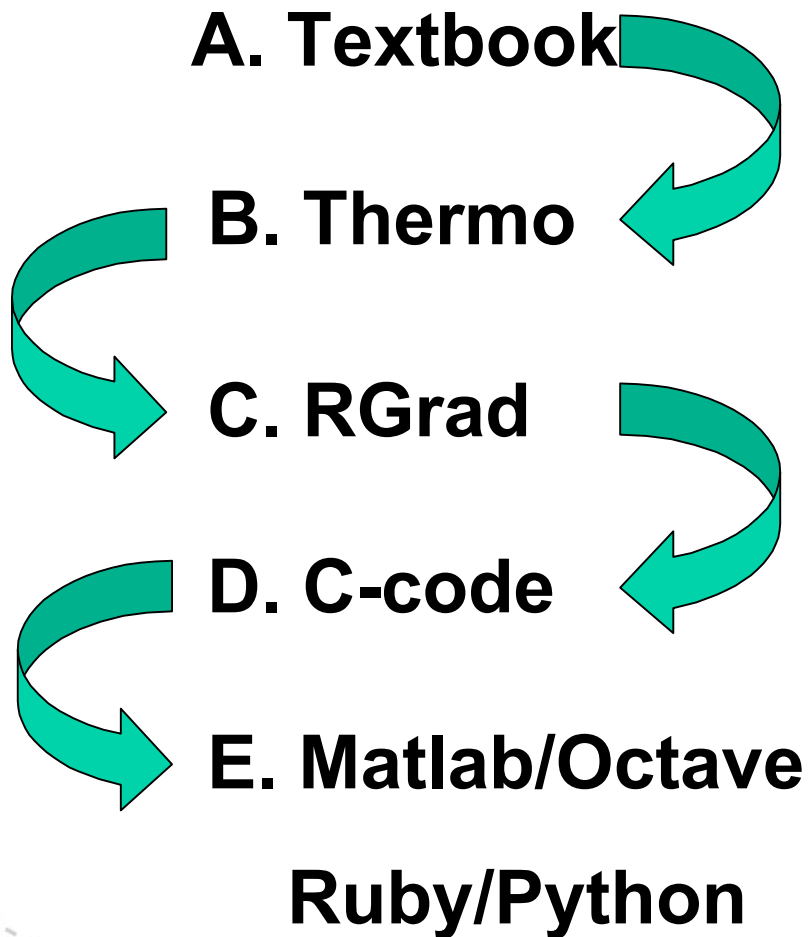
Tailor-made **C**-code without dependencies (no libraries or licence issues) using a built-in program interface to store and retrieve model parameters, state variables and calculated properties.

Utility **D**ata wrapper for effective multi-dimensional array handling, ie. each element of the array [c1, c2, ...] will be a model instance from **C** running within its own memory

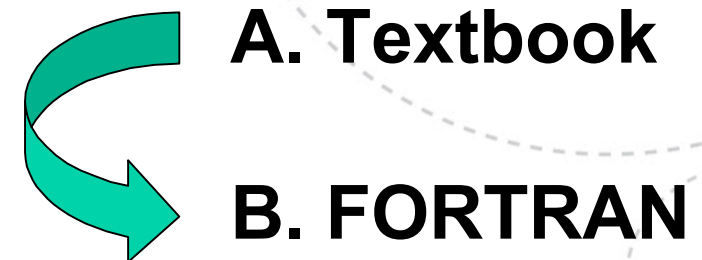
Equation handler for mapping the elements of [c1, c2, ...] into a topology graph (takes the form of an incidence matrix)

Flowsheet solver

Open source



Traditional



Step A: Textbook example

Thermodynamic functions are - more often than not - quite complex with double, or even triple, summations over the components. The E-NRTL model can serve as an example:

$$\begin{aligned} \frac{g^{ex,lc}}{RT} = & \sum_m X_m \frac{\sum_j X_j G_{jm} \tau_{jm}}{\sum_k X_k G_{km}} \\ & + \sum_c X_c \sum_{a'} \frac{X_{a'}}{\sum_{a''} X_{a''}} \frac{\sum_j X_j G_{jc,a'} \tau_{jc,a'}}{\sum_k X_k G_{kc,a'}} \\ & + \sum_a X_a \sum_{c'} \frac{X_{c'}}{\sum_{c''} X_{c''}} \frac{\sum_j X_j G_{ja,c'} \tau_{ja,c'}}{\sum_k X_k G_{ka,c'}} \end{aligned}$$

Step B1: Thermo (language)

```

cp   = MuT_cp::new           # Cp(T)
h    = MuT_hs::new          # Standard state
s    = MuT_hs::new          # Standard state
id   = ModTN_ideal::new     # Ideal mixture
gex  = ModTN::new           # Excess Gibbs energy

```

Phase tag

Function

Database

Location

```

cp[LIQ] = :poly3, REID87, HOME
h[LIQ]  = :h0, DIPPR96, HOME   # Heat of formation
s[LIQ]  = :s0, DIPPR96, HOME   # Standard entropy
id[LIQ] = :ionicmix           # Ideal electrolyte
gex[LIQ] = :e-nrtl, ASPEN, USR # Electrolyte NRTL

```

Step B2: Thermo (algebra)

Component list:

$n = [\text{'water'}, \text{'HNO3'}, \text{'H<+>'}, \text{'K<+>'}, \text{'NO3<->'}, \text{'Cl<->'}]$

Thermo model:

$g = \text{Surface} * (\text{Gibbs} * (\text{StandardState} * (\text{cp} * (\text{h} + \text{s.mixture}(n))) + \text{Activity} * \text{id} + \text{Activity} * \text{gex.mixture}(n))$

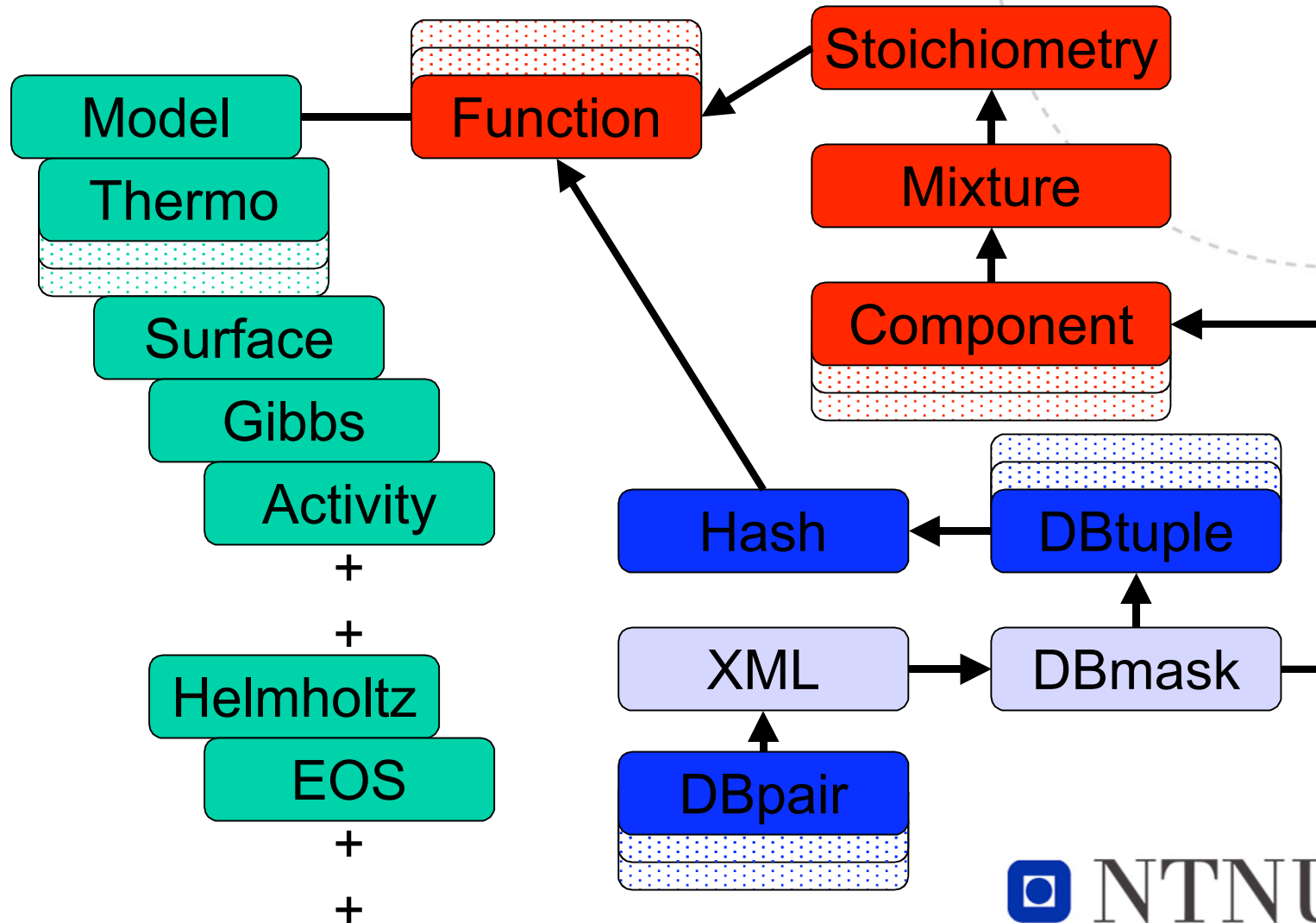
Available EOS

- Ideal gas
- Second virial EOS
- VdW
- RK, SRK, PR
- VTSRK, VTPR (volume translated)
- Saul-Wagner polar fluids: NH₃, etc.
- Saul-Wagner non-polar fluids: CO₂, etc.
- Span-Wagner water and steam: 38 and 58 param.
- RK-ASPEN (Mathias)
- Schwartzenuber-Renon-Watanasiri

Available activity models

- Ideal mixture
- Wilson, NRTL
- Redlich-Kister (N-degree)
- ASPEN E-NRTL

Thermo (architecture)



Step C: RGrad (language)

t_{ijk} are interaction parameters

g_{ijk} are $\exp(-(a_{ijk} * t_{ijk})/RT)$

N , N_a , N_c are number of species, anions and cations.

m , m_a , m_c are moles of species, anions and cations.

```
zik = RGrad::expr(m[nil,Nc,nil],c[nil,Nc,nil],gijk,tijk){|_mj,_cj
    _tijk*_gijk*_mj*_cj
  }.sum!(Na,nil,Nc)
```

Energy parameter

```
gi = RGrad::expr(mc[nil,Nc],bik,zik){|_mck,_bik,_zik|
    _mck*_zik/_bik
  }.sum!(Na,nil)
```

Energy contribution

```
gex = RGrad::expr(ya,gi){|_yai,_gi|
    _yai*_gi
  }.sum!
```

Cation excess Gibss



NTNU

Innovation and Creativity

Step C: RGrad (language)

Gradient calculations are easy in RGrad (which by the

way is an acronym for RubyGradients):

```
dgdt = g.grad(t)
dgdp = g.grad(p)
dgdn = g.grad(n)
dgdtt = dgdt.grad(t)
dgdtp = dgdt.grad(p)
```

Up to 6th order has been tested

Legendre transforms are also possible:

```
u = g.legendre(t,p,n).legendre(p,t,n)
duds = u.grad(t)
dudv = u.grad(p)
dudn = u.grad(n)
```

Transformed variables

Step D: C-code

pandora.h
pandora.c

100 lines of declarations.

700 lines of multidimensional array calculations.

rgradapi.h

20 lines of generic API.

e_nrtl.c

1400 lines of model specific code.

rgrad_call.c

&

rgrad_add_to_values.c

&

rgrad_set.c

&

rgrad_get.c, rgrad_free.c

100 lines of generic API.

e_nrtl_init.c

600 lines of MEX interface

e_nrtl_rb_interface.c

500 lines of Ruby interace.

Step D: C-code

```

d0          = makeDim(sized0, arrd0);
int u15[6]  = {0, 0, 0, 0, 0, 0};
storage->p[20] = makePandora(6, u15, 3, 42, D2D0D3,
                             p20, d2, d0, d3);
storage->e[20] = allocPandora(6, u15, 3, d2, d0, d3);

niter(e[20], lambda2, 3, elem, tmp2, tmp1,
      tfarr, p[20], ttt, p[21], ttt, e[0], fff);

static double lambda2(double *a){
    return exp(-(a[0] * a[1]) / 0.083145119843087 / a[2]);
}

```

Annotations:

- Array dimension (points to `sized0`)
- Scientific unit (points to `arrd0`)
- gijk parameter (points to `u15`)
- gijk expression (points to `6`)
- gijk memory (points to `u15`)
- gijk function (points to `lambda2`)

Thermo > RGrad > C summary

- A. The thermodynamic modelling takes place at a high level of abstraction using the Thermo-library (10-30 lines of code).
- B. The model structure is checked for consistency, function definitions are loaded, databases are searched, and scientific units are checked as the model is compiled into RGrad code (1000 lines).
- C. The RGrad model is differentiated and compiled into C-code (1000-100,000 lines).
- D. The C-code is compiled and linked into a dynamic linked library (Windows), shared object (Linux) or bundle (Mac).

Step E: Applications (critical point)

1. Homogeneity of first order means that the Hessian of A has a zero curvature in the direction of the (extensive) state vector (V, N) .
2. Criticality means that a second eigenvalue will approach zero.
3. The criticality condition can be extended to any of the Legendre transforms of A , including:

$$X(-S, V, \mu) = \Omega(T, V, \mu) - \Omega \frac{\partial \Omega}{\partial V} V$$

$$= A(T, V, N) - A \frac{\partial A}{\partial V} V + \sum_i \frac{\partial A}{\partial N_i} N_i$$

4. Note: Only 2 variables, even in multicomponent systems.

Criticality

Not only does the Hessian vanish, but the 3rd derivative must also be zero (for a stable critical point). We may therefore write:

$$X_{VV} = 0$$

$$X_{VVV} = 0$$

$$\text{However, } X_{VV} = -\begin{pmatrix} X_{VVS} & X_{VVV} \end{pmatrix} \begin{pmatrix} -S \\ V \end{pmatrix}$$

which means both $X_{VVV} = 0$ and $X_{VVS} = 0$

Actually, these are the conditions solved for in order to locate the critical state.

Criticality (cont'd)

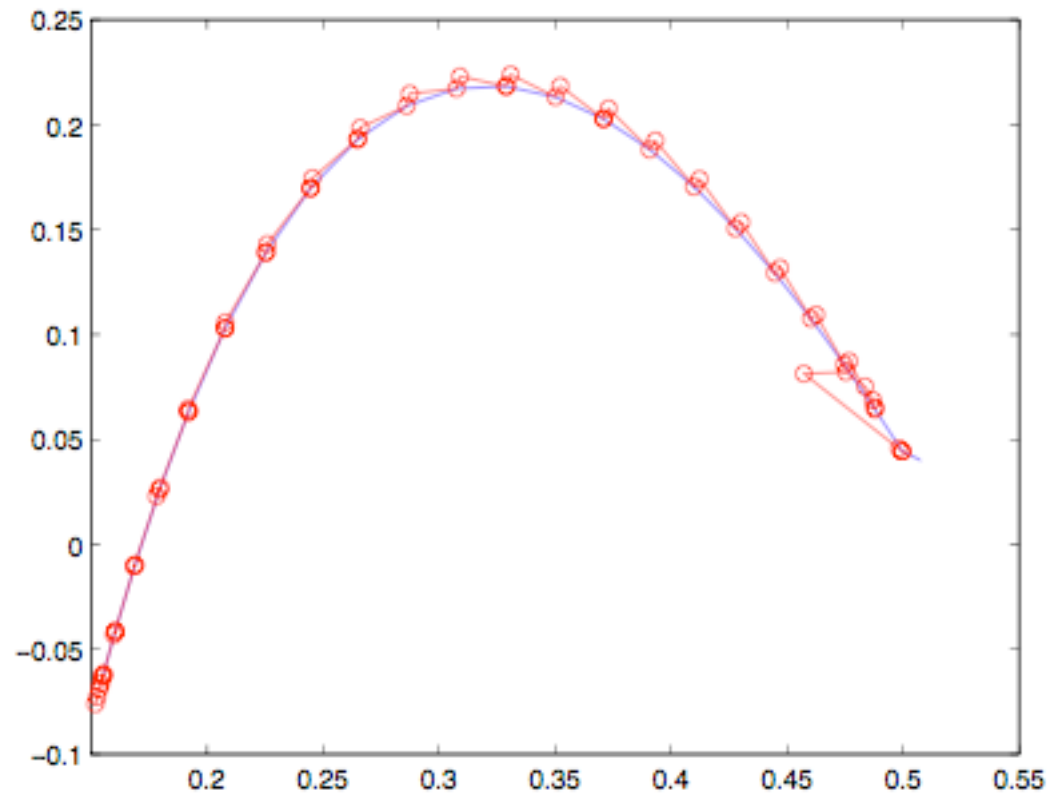
However, there are numerical caveats lurking in the water. From the 4th order homogeneity we can write:

$$\begin{pmatrix} X_{VVSS} & X_{VVS V} \\ X_{VVVS} & X_{VVVV} \end{pmatrix} \begin{pmatrix} -S \\ V \end{pmatrix} = -2 \begin{pmatrix} X_{VVS} \\ X_{VVV} \end{pmatrix}$$

But since the right hand side vanishes at the critical point it also means that the coefficient matrix (which is actually the Hessian of X_{VV}) becomes singular at the critical point. This turns the Newton method into a non-Banach fix point iterator. Strange!

Numerical results

T,P critical phase locus for C1-C6 (temperatures in 1000K and pressures in 0.01MPa):



Concluding remarks

1. The Thermo software has been (re)written to take advantage of automatic differentiation with respect to state variables and model parameters.
2. Compilation from Thermo to RGrad has been tested for a wide range of Helmholtz and Gibbs energy models.
3. All RGrad parameters and expressions have scientific units checking. Inconsistent models will not compile.
4. Compilation from RGrad to C makes code which competes with Matlab in speed.
5. The C-code can be executed from both Matlab/Octave and Ruby (Python is under development).