```
> restart;
```
VLE data for chloroform(1)/1,4-dioxane(2) at 50 degC (S,vN&A Table 11.3)

Utility routines

```
> with(linalg): with(plots):with(stats):
Warning, new definition for norm
Warning, new definition for trace
```

Trapezoid-rule integration of a data set

x and y are lists describing the data; i0 and i1 indicate the lower and upper points through which the integration is performed

```
> trapezoid := (i0,i1,x,y) ->
  sum(0.5*(y['i'+1]+y['i'])*(x['i'+1]-x['i']),'i'=i0..i1-1);
```

$$trapezoid := (i0, i1, x, y) \rightarrow \sum_{'i' = i0}^{i1-1} (.5\,(y_{'i'+1} + y_{'i'})\,(x_{'i'+1} - x_{'i'}))$$

Routine to plot a set of data

x and y are lists describing the data; i0 and i1 indicate the lower and upper points through which the plot is made

```
> plotdata := (i0,i1,x,y) -> plot([[x['i'],y['i']]
  $'i'=i0..i1],color=black,style=point,symbol=circle):
```

Very crude minimization routine

Finds rough minimum of a function of two variables. Examines value of function over an equally-spaced grid of values of the two variables. "fn" is the function, which should take two arguments; "nPts" is the number of grid points for each variable; "x1Range" and "x2Range" are lists (of the form [x1min,x1max]) which specify the upper and lower bounds of the grid for each variable. You can refine the search by running the routine several times, each one with a narrower range of values for the two variables.

```
> crudeMinimize := proc(fn,nPts,x1Range,x2Range)
     local x1,x2,m1,m2,x1step,x2step,obj,bestobj,x1best,x2best;
     x1step := (x1Range[2]-x1Range[1])/nPts;
     x2step := (x2Range[2]-x2Range[1])/nPts;
     bestobj := 1e32;
     for m1 from 0 to nPts do
       x1 := x1Range[1] + m1*x1step;
       for m2 from 0 to nPts do
         x2 := x2Range[1] + m2*x2step;
         obj := fn(x1,x2);
         if(obj < bestobj) then
            bestobj := obj;
            x1best := x1;
            x2best := x2;
          fi;
        od;
     od;
     evalf([x1best,x2best,bestobj]);
  end:
```

Gibbs-Excess models

All models are written to take three arguments: the mole fraction of component 1, and two modeling parameters (for example, A12 and A21 for the 2-parameter Margules equation). Some models require fewer parameters (or none); for these the extra arguments just fill a space and are not used. The routines are written this way so that they can be used interchangably in the dew- and bubble-point programs below. (Improved versions are being developed that permit treatment of multicomponent mixtures, and which do not require passing of dummy parameters).

Ideal solution

```
> gEIdeal := (x,dummy1,dummy2) -> 0.0:
  gamma1Ideal := (x,dummy1,dummy2) -> 1.0:
  gamma2Ideal := (x,dummy1,dummy2) -> 1.0:
```

1-constant Margules

```
> gEMargules1 := (x,A,dummy) -> A*x*(1-x):
  gamma1Margules1 := (x,A,dummy) -> exp(A*(1-x)^2):
  gamma2Margules1 := (x,A,dummy) -> exp(A*x^2):
```

2-constant Margules

```
> gEMargules2 := (x,A12,A21) -> (A21*x + A12*(1-x))*x*(1-x):
  gamma1Margules2 := (x,A12,A21) -> exp((1-x)^2 * (A12 +
  2*(A21-A12)*x)):
  gamma2Margules2 := (x,A12,A21) -> exp(x^2 * (A21 +
  2*(A12-A21)*(1-x))):
```

van Laar

```
> gEvanLaar := (x,A12p,A21p) -> A12p*A21p/(A12p*x +
  A21p*(1-x))*x*(1-x):
> gamma1vanLaar := (x,A12p,A21p) -> exp(A12p/(1 +
  A12p*x/A21p/(1-x))^2):
  gamma2vanLaar := (x,A12p,A21p) -> exp(A21p/(1 +
  A21p*(1-x)/A12p/x)^2):
```

Wilson

```
> gEWilson := (x,L12,L21) -> -x*ln(x + (1-x)*L12) -
  (1-x)*ln((1-x) + x*L21):
  gamma1Wilson := (x,L12,L21) -> exp(-ln(x + (1-x)*L12) +
  (1-x)*(L12/(x + (1-x)*L12) - L21/((1-x) + x*L21))):
  gamma2Wilson := (x,L12,L21) -> exp(-ln((1-x) + x*L21) -
  x*(L12/(x + (1-x)*L12) - L21/((1-x) + x*L21))):
```

The following routine is under development. It will be applicable to multicomponent (not just binary) mixtures.

```
gammaWilson := proc(n,T,x,C)
   local V, aR, L, i, j;
   V := C[1];
   aR := C[2];
print(V);
print(aR);
   L :=
[seq(seq([V[j]/V[i]*exp(-aR[i,j]/T)],j=1..n),i=1..n)];
#print(L);
#   [evalf(seq( 1 - ln(sum(x['j']*L[i,'j'],'j'=1..n)) -
sum(x[i]*L['k',i]/sum(x['j']*L['k','j'],'j'=1..n),'k'=1..n),i
=1..n))];
print(x);
   [evalf(seq( 1 - ln(sum(x['j']*L['i','j'],'j'=1..n))
,'i'=1..n))];
```

```
  end:
```

Dew- and bubble-point calculation routines

Antoine equation and its inverse

```
> pSat := (T,A,B,C) -> evalf(exp(A - B/(T+C))):
  TSat := (P,A,B,C) -> evalf(B/(A - ln(P)) - C):
```

Antoine constants for some substances (for T in degC)

```
> ABCacetonitrile := [14.2724,2945.47,224.0]:
  ABCnitromethane := [14.2043,2972.64,209.0]:
  ABC2propanol := [16.6780,3640.20,273.15-53.54]:
  ABCwater := [16.2887,3816.44,273.15-46.13]:
```

Bubble pressure routine

Returns the bubble pressure and vapor mole fraction for a binary mixture. Assumes ideal gases and unit Poynting correction. Follows algorithm described by Fig. 12.12 of SvN&A.
This routine takes the following parameters:
  T = temperature in degrees Celsius
  x = liquid mole fraction of species 1 (of a two-component mixture)
  ABC1, ABC2 = lists of Antoine-equation constants (of the form [A, B, C]) for components 1 and 2, respectively
  gamma1Model, gamma2Model = name of functions that return the respective activity coefficients for species 1 and 2. These routines should take three arguments, as described in the "Gibbs-excess models" above
  gammaCoeffs = list of two coefficients that are passed to the functions "gamma1Model" and "gamma2Model"

```
> BubblePressure :=
  proc(T,x,ABC1,ABC2,gamma1Model,gamma2Model,gammaCoeffs)
    local p1Sat, p2Sat, gamma1, gamma2, P, y;
    p1Sat := pSat(T,ABC1[1],ABC1[2],ABC1[3]);
    p2Sat := pSat(T,ABC2[1],ABC2[2],ABC2[3]);
    gamma1 := gamma1Model(x,gammaCoeffs[1],gammaCoeffs[2]);
    gamma2 := gamma2Model(x,gammaCoeffs[1],gammaCoeffs[2]);
    P := x*gamma1*p1Sat + (1-x)*gamma2*p2Sat;
    y := x*gamma1*p1Sat/P;
    [P,y];
  end:
```

Reproduce result from Example 12.1, page 444. Uses ideal-solution model for liquid.

```
> BubblePressure(75,0.2,ABCacetonitrile,ABCnitromethane,gamma1Ide
  al,gamma2Ideal,[dummy1,dummy2]);
                        [50.22753556, .3313196896]
```

Dew temperature routines

2-component

Returns the dew temperature and liquid mole fraction for a binary mixture. Assumes ideal gases and unit Poynting correction. Follows algorithm described by Fig 12.15 of SvN&A.
This routine takes the following parameters:
  P = pressure in units consistent with Antoine-equation vapor pressure (kPa for the examples used here)
  y = vapor mole fraction of species 1 (of a two-component mixture)
  ABC1, ABC2 = lists of Antoine-equation constants (of the form [A, B, C]) for components 1 and 2, respectively
  gamma1Model, gamma2Model = name of functions that return the respective activity

coefficients for species 1 and 2. These routines should take three arguments, as described in the "Gibbs-excess models" above
gammaCoeffs = list of two coefficients that are passed to the functions "gamma1Model" and "gamma2Model"

```
> DewTemperature :=
  proc(P,y,ABC1,ABC2,gamma1Model,gamma2Model,gammaCoeffs)
    local T1Sat, T2Sat, p1Sat, p2Sat, gamma1, gamma2, x1,
  x2,  gammaOld, xsum, T, TOld, xi, epsilon, dgamma, dT;
    xi := 1.0e-4; epsilon := 1.0e-4;    xi and epsilon are the
  convergence tolerances for the iteration loops
    dT := 1e32;    dT is the temperature change from one iteration to the next.
  When it is less than epsilon, convergence is reached.  Initialize it here to a large value.
    gamma1 := 1;  gamma2 := 1;    Activity coefficients of two species
    T1Sat := TSat(P,ABC1[1],ABC1[2],ABC1[3]);    Compute saturation
  temperatures at P, according to algorithm
    T2Sat := TSat(P,ABC2[1],ABC2[2],ABC2[3]);
    T := y*T1Sat + (1-y)*T2Sat;    Initial guess of dew temperature
    p1Sat := pSat(T,ABC1[1],ABC1[2],ABC1[3]);    Compute saturation
  pressures at guessed dew T, according to algorithm
    p2Sat := pSat(T,ABC2[1],ABC2[2],ABC2[3]);
    p1Sat := P*(y/gamma1 + (1-y)/gamma2*p1Sat/p2Sat);   Choose
  component 1 as "species j" of algorithm
    T := TSat(p1Sat,ABC1[1],ABC1[2],ABC1[3]);
    p2Sat := pSat(T,ABC2[1],ABC2[2],ABC2[3]);
    x1 := y*P/gamma1/p1Sat;        initial guess of liquid mole fraction
    x2 := (1-y)*P/gamma2/p2Sat;
    gamma1 :=
  evalf(gamma1Model(x1,gammaCoeffs[1],gammaCoeffs[2]));
    gamma2 :=
  evalf(gamma2Model(x1,gammaCoeffs[1],gammaCoeffs[2]));
    p1Sat := P*(y/gamma1 + (1-y)/gamma2*p1Sat/p2Sat);
    T := TSat(p1Sat,ABC1[1],ABC1[2],ABC1[3]);
    while dT > epsilon do    loop until temperature change is less than
  epsilon
       TOld := T;
       dgamma := [1e32,1e32];    dgamma is a list showing the change in
  gamma1 and gamma2 on successive iterations of the inner  loop
       p1Sat := pSat(T,ABC1[1],ABC1[2],ABC1[3]);
       p2Sat := pSat(T,ABC2[1],ABC2[2],ABC2[3]);
       while max(op(dgamma)) > xi do loop until the largest element of
  dgamma is less than xi.  (the op function takes the list and returns a sequence, i.e., it
  takes away the square brackets, so the proper format for the max function is presented)

          gammaOld := [gamma1,gamma2];
          x1 := y*P/gamma1/p1Sat;
          x2 := (1-y)*P/gamma2/p2Sat;
          xsum := x1 + x2;
          x1 := x1/xsum; x2 := x2/xsum;
          gamma1 :=
  evalf(gamma1Model(x1,gammaCoeffs[1],gammaCoeffs[2]));
          gamma2 :=
  evalf(gamma2Model(x1,gammaCoeffs[1],gammaCoeffs[2]));
          dgamma := map(abs,[gamma1,gamma2] - gammaOld);
  this takes the absolute value of each gamma change and makes a list of them
```

```
#         print(dgamma);
```
remove hash mark at beginning of line to observe convergence of gamma
```
      od;
      p1Sat := P*(y/gamma1 + (1-y)/gamma2*p1Sat/p2Sat);
      T := TSat(p1Sat,ABC1[1],ABC1[2],ABC1[3]);
      dT := abs(T - TOld);
```
we want absolute value of temperature change, so that a large negative dT is not interpreted as converged
```
#     print(dT);
```
remove hash mark at beginning of line to observe convergence of temperature
```
   od;
   [T,x1];
end:
```

Multicomponent (under development)

Again reproduce result from Example 12.1.

```
> DewTemperature(50.23,.3313,ABCacetonitrile,ABCnitromethane,gamma1Ideal,gamma2Ideal,[dummy1,dummy2]);
```

$$[75.0016825, .1999868410]$$

```
> DewTemperature(20,.5,ABCacetonitrile,ABCnitromethane,gamma1Margules1,gamma2Margules1,[-1.0,dummy2]);
```

$$[54.1499219, .3727474432]$$

Dew temperature using Maple's solve routine!

```
> DewTemperature2 :=
  (P,y,ABC1,ABC2,gamma1Model,gamma2Model,gammaCoeffs) ->

  fsolve({x*gamma1Model(x,gammaCoeffs[1],gammaCoeffs[2])*pSat(T,ABC1[1],ABC1[2],ABC1[3])=y*P,

  (1-x)*gamma2Model(x,gammaCoeffs[1],gammaCoeffs[2])*pSat(T,ABC2[1],ABC2[2],ABC2[3])=(1-y)*P},{T,x},{x=0..1,T=1..1000}):
> DewTemperature2(20,.5,ABCacetonitrile,ABCnitromethane,gamma1Margules1,gamma2Margules1,[-1.0,dummy2]);
```

$$\{ T = 54.14992374, x = .3727449080 \}$$

Routine to test results by comparing liquid and vapor fugacities.

```
> DewTemperatureTest :=
  (x,T,P,y,ABC1,ABC2,gamma1Model,gamma2Model,gammaCoeffs) ->

  [x*gamma1Model(x,gammaCoeffs[1],gammaCoeffs[2])*pSat(T,ABC1[1],ABC1[2],ABC1[3])=y*P,

  (1-x)*gamma2Model(x,gammaCoeffs[1],gammaCoeffs[2])*pSat(T,ABC2[1],ABC2[2],ABC2[3])=(1-y)*P]:
> DewTemperatureTest(.3727,54.15,20,.5,ABCacetonitrile,ABCnitromethane,gamma1Margules1,gamma2Margules1,[-1.0,dummy2]);
```

$$[9.998260897 = 10.0, 10.00108349 = 10.0]$$

```
>
```