The Lennard-Jones Equation of State

David Keffer
Department of Chemical Engineering
University of Tennessee, Knoxville
Begun: February 5, 2002
Last Updated: February 5, 2002

Table of Contents

When you use microcanonical simulations, you specify N, V, and T. If you would like to simulate at a given pressure, you need to run in the Gibbs ensemble and specify N, P, and T (which requires that you write a Gibbs ensemble MD code). A simple but less rigorous alternative is to use an equation of state to predict the density corresponding to a particular pressure, then simulate with the microcanonical code at that density. To this end, one should use the Lennard-Jones equation of state.[1]

This is an excellent estimate for the gas phase. If we want the density of a liquid at, for example, atmospheric pressure, then using this technique, one can get a pretty close estimate of the density. One can always compute the pressure during the simulation to determine how close the estimate was. Practically speaking, the standard deviation of the pressure fluctuations during a molecular dynamics simulation of a liquid can be 10 atms, making it difficult to verify whether the average pressure is really 1 atm.

Nevertheless, this technique is a simple and practical way to find a density for a given pressure, in order to run a microcanonical ensemble.

The four appendices that follow provide four FORTRAN 90 codes that return either pressure given temperature and molar volume or return molar volume given temperature and pressure. The second task requires a Newton-Raphson convergence scheme. The tasks are performed both the pure fluid as well as for a mixture with an arbitrary number of components.

1. lj_eos_pure.for
   system: pure fluid
   input: σ, ε, temperature, molar volume
   output: pressure

2. lj_eos_pure_TP.for
   system: pure fluid
   input: σ, ε, temperature, pressure
   output: molar volume

3. lj_eos_mix.for
   system: mixture
   input: $\underline{\sigma}$, $\underline{\varepsilon}$, mole fractions, temperature, molar volume
   output: pressure

4. lj_eos_mix_TP.for
   system: mixture
   input: $\underline{\sigma}$, $\underline{\varepsilon}$, mole fractions, temperature, pressure
   output: molar volume

The mixture rules for a mixture with n_c components used in the last two FORTRAN codes are given below. They are standard mixing rules for equations of state.[2] The mean collision diameter for the mixture is the sum of the pure component collision diameters, weighted by the mole fractions.

$$\sigma = \sum_{i=1}^{n_c} \sigma_i x_i \tag{1}$$

The mean well depth for the mixture is defined as follows.

$$\varepsilon = \sum_{i=1}^{n_c} \sum_{j=1}^{n_c} \varepsilon_{ji} x_i x_j \tag{2}$$

$$\varepsilon_{ij} = \left(1 - k_{ij}\right)\sqrt{\varepsilon_i \varepsilon_j} \tag{3}$$

where k_{ij} is the binary interaction parameter. The binary interaction parameter is zero for ideal mixtures. Thus $k_{ii} = 0$ for all i. The matrix of binary interaction parameters is symmetric. Thus $k_{ij} = k_{ji}$ for all i and j. In a binary mixture, you need one interaction parameter. In a ternary mixture, you need 3. In a four-component mixture, you need 6.

References:

1. Nicolas, J.J., Gubbins, K.E., Streett, W.B., Tildesley, D.J., "Equation Of State For The Lennard-Jones Fluid", *Mol. Phys.* **37**(5) 1979 p.1429-1454.
2. Sandler, S.I., Chemical and Engineering Thermodynamics, John Wiley & Sons, New York, 1989, p. 318.

Appendix A. FORTRAN Code for Lennard-Jones EOS, pure fluid, given ρ, find P

```
      program lj_eos_pure
c     This program calculates the pressure using the equation of state
c     for the Lennard Jones fluid
c     Ref : Nicolas et. al. , "Molecular Physics, 1979, vol. 37, No. 5
c     pp 1429 - 1454.
c     Author : Parag Adhangale
c     Date created : March 10, 00.
c     Date last modified : Jan 26, 2002.
      implicit double precision (a-h,o-z)
      double precision, parameter :: sig = 5.8810d0 ! sigma (Angstroms)
      double precision, parameter :: eps = 327.0d0 ! eps (K)
      double precision, parameter :: T = 300.0d0 ! T (K)
      double precision, parameter :: vn = 275.0d0 ! molar volume (A^3/molecule)
      double precision, parameter :: xk = 1.38066e-23 ! Boltzmann's Constant
      double precision, dimension(1:32) :: c
      rho = 1.d0/vn
c
c  Constants for the LJ eos :
c
      gam = 3.000000e+00
      c (1) = -0.44480725e-01
      c (2) = 00.72738221e+01
      c (3) = -0.14343368e+02
      c (4) = 00.38397096e+01
      c (5) = -0.20057745e+01
      c (6) = 00.19084472e+01
      c (7) = -0.57441787e+01
      c (8) = 00.25110073e+02
      c (9) = -0.45232787e+04
      c (10)= 00.89327162e-02
      c (11)= 00.98163358e+01
      c (12)= -0.61434572e+02
      c (13)= 00.14161454e+02
      c (14)= 00.43353841e+02
      c (15)= 00.11078327e+04
      c (16)= -0.35429519e+02
      c (17)= 00.10591298e+02
      c (18)= 00.49770046e+03
      c (19)= -0.35338542e+03
      c (20)= 00.45036093e+04
      c (21)= 00.77805296e+01
      c (22)= 00.13567114e+05
      c (23)= -0.85818023e+01
      c (24)= 00.16646578e+05
      c (25)= -0.14092234e+02
      c (26)= 00.19386911e+05
      c (27)= 00.38585868e+02
      c (28)= 00.33800371e+04
      c (29)= -0.18567754e+03
      c (30)= 00.84874693e+04
      c (31)= 00.97508689e+02
      c (32)= -0.14483060e+02
c     calculate reduced T, rho, and pres
```

```
      rt = T / eps
      rr = rho * sig**3.d0;
      rpres = rr * rt + rr**2.d0 * (c(1) * rt + c(2)*rt**(1.d0/2.d0)
     &     + c(3) +  c(4)*rt**(-1.d0) + c(5)*rt**(-2.d0))
     &     +  rr**3.d0*(c(6)*rt + c(7) + c(8)*rt**(-1.d0)
     &     + c(9)*rt**(-2.d0))
     &     +  rr**4.d0*(c(10)*rt + c(11) + c(12)*rt**(-1.d0))
     &     + rr**5.d0*c(13)
     &     +  rr**6.d0*(c(14)*rt**(-1.d0) + c(15)*rt**(-2.d0))
     &     +  rr**7.d0*(c(16)*rt**(-1.d0)) + rr**8.d0*(c(17)*rt**(-1.d0)
     &     +  c(18)*rt**(-2.d0)) + rr**9.d0*(c(19)*rt**(-2.d0))
     &     +  rr**3.d0*(c(20)*rt**(-2.d0) + c(21)*rt**(-3.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**5.d0*(c(22)*rt**(-2.d0) + c(23)*rt**(-4.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**7.d0*(c(24)*rt**(-2.d0) + c(25)*rt**(-3.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**9.d0*(c(26)*rt**(-2.d0) + c(27)*rt**(-4.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**11.d0*(c(28)*rt**(-2.d0) + c(29)*rt**(-3.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**13.d0*(c(30)*rt**(-2.d0) + c(31)*rt**(-3.d0)
     &     + c(32)*rt**(-4.d0))
     &     *dexp(-gam*rr**2.d0)
c          put some units on pres (aJ/molecule)
      pres_pa = rpres * eps * xk / sig**3.d0 / 1.0e-30;
c          convert from Pa to bar
           pres_bar = pres_pa*1.0d-5;
           print *, ' sigma = ', sig, ' Angstroms & epsilon = ', eps, ' K '
           print *, ' T = ', T, ' K & Vn = ', vn, ' Angstroms^3/molecule '
           print *, 'Pressure = ', pres_pa, ' Pa = ', pres_bar, ' bar '
           stop
           end
```

Appendix B. FORTRAN Code for Lennard-Jones EOS, pure fluid, given P, find ρ

```
      program lj_eos_pure_TP
c     This program uses a Newton Raphson method
c     with numerical derivatives to solve for
c     the molar volume, given temperature and pressure,
c     for the Lennard Jones Equation of State
c
c     Author:  David Keffer
c     Begun:   02/05/02
c     completed:  02/05/02
c
      implicit double precision (a-h, o-z)
      double precision, parameter :: sig = 5.8810d0 ! sigma (Angstroms)
      double precision, parameter :: eps = 327.0d0 ! eps (K)
      double precision, parameter :: T = 300.0d0 ! T (K)
      double precision, parameter :: pres_targ = 101325.0d0 ! Pressure (Pa)
c     set the initial guess for the molar volume (A^3/molecule)
      vn_old = 275.d0
c
c Newton-Raphson method with numerical approximations to the derivative.
c

      maxit = 100;
      tol = 1.0d-8;
      err = 100.d0;
      icount = 0;
      xold = vn_old;
      do while (err .gt. tol .and. icount .le. maxit)
            icount = icount + 1;
            call funkeval(sig, eps, T, xold, f, pres_targ)
            print *, ' i= ', icount, ' vn= ', xold, ' f= ', f
            h = min(0.01*xold,0.01)
            call dfunkeval(xold, h, sig, eps, T, df, pres_targ)
            xnew = xold - f/df
            if (icount .gt. 1) then
                  err = abs((xnew - xold)/xnew)
            endif
            xold = xnew
            print *, ' i= ', icount, ' vn= ', xold, ' err= ', err
      enddo
      x0 = xnew
      if (icount .ge. maxit) then
c           you ran out of iterations
            print *, 'Sorry. You did not converge in ',maxit,' iterations'
            print *, 'The final value of x was ', x0
      endif
      vn = x0
      pres_pa = pres_targ
      pres_bar = pres_pa*1.0d-5;
      print *, ' sigma = ', sig, ' Angstroms & epsilon = ', eps, ' K '
      print *, ' T = ', T, ' K & Vn = ', vn, ' Angstroms^3/molecule '
      print *, 'Pressure = ', pres_pa, ' Pa = ', pres_bar, ' bar '
      stop
```

```
      end

c
c     This subroutine calculates a numerical approximation to the derivative
c
      subroutine dfunkeval(vn, h, sig, eps, T, df, pres_targ)
      implicit double precision (a-h,o-z)
      double precision, intent(in) :: sig, eps, T, vn, h, pres_targ
      double precision, intent(out) :: df
      call funkeval(sig, eps, T, vn+h, fp, pres_targ)
      call funkeval(sig, eps, T, vn-h, fn, pres_targ)
      df = (fp - fn)/(2*h);
      return
      end

c
c     This subroutine evaluates the function
c
      subroutine funkeval(sig, eps, T, vn, fobj, pres_targ)
c     This program calculates the pressure using the equation of state
c     for the Lennard Jones fluid
c     Ref : Nicolas et. al. , "Molecular Physics, 1979, vol. 37, No. 5
c    pp 1429 - 1454.
c     Author : Parag Adhangale
c     Date created : March 10, 00.
c     Date last modified : Jan 26, 2002.
      implicit double precision (a-h,o-z)
      double precision, intent(in) :: sig, eps, T, vn, pres_targ
      double precision, intent(out) :: fobj
      double precision, parameter :: xk = 1.38066e-23 ! Boltzmann's Constant
      double precision, dimension(1:32) :: c
      rho = 1.d0/vn
c
c  Constants for the LJ eos :
c
    gam = 3.000000e+00
    c (1) = -0.44480725e-01
    c (2) = 00.72738221e+01
    c (3) = -0.14343368e+02
    c (4) = 00.38397096e+01
    c (5) = -0.20057745e+01
    c (6) = 00.19084472e+01
    c (7) = -0.57441787e+01
    c (8) = 00.25110073e+02
    c (9) = -0.45232787e+04
    c (10)= 00.89327162e-02
    c (11)= 00.98163358e+01
    c (12)= -0.61434572e+02
    c (13)= 00.14161454e+02
    c (14)= 00.43353841e+02
    c (15)= 00.11078327e+04
    c (16)= -0.35429519e+02
    c (17)= 00.10591298e+02
    c (18)= 00.49770046e+03
    c (19)= -0.35338542e+03
    c (20)= 00.45036093e+04
```

```
      c (21)= 00.77805296e+01
      c (22)= 00.13567114e+05
      c (23)= -0.85818023e+01
      c (24)= 00.16646578e+05
      c (25)= -0.14092234e+02
      c (26)= 00.19386911e+05
      c (27)= 00.38585868e+02
      c (28)= 00.33800371e+04
      c (29)= -0.18567754e+03
      c (30)= 00.84874693e+04
      c (31)= 00.97508689e+02
      c (32)= -0.14483060e+02
c           calculate reduced T, rho, and pres
      rt = T / eps
      rr = rho * sig**3.d0;
      rpres = rr * rt + rr**2.d0 * (c(1) * rt + c(2)*rt**(1.d0/2.d0)
     &     + c(3) +  c(4)*rt**(-1.d0) + c(5)*rt**(-2.d0))
     &     +  rr**3.d0*(c(6)*rt + c(7) + c(8)*rt**(-1.d0)
     &     + c(9)*rt**(-2.d0))
     &     +  rr**4.d0*(c(10)*rt + c(11) + c(12)*rt**(-1.d0))
     &     + rr**5.d0*c(13)
     &     +  rr**6.d0*(c(14)*rt**(-1.d0) + c(15)*rt**(-2.d0))
     &     +  rr**7.d0*(c(16)*rt**(-1.d0)) + rr**8.d0*(c(17)*rt**(-1.d0)
     &     +  c(18)*rt**(-2.d0)) + rr**9.d0*(c(19)*rt**(-2.d0))
     &     +  rr**3.d0*(c(20)*rt**(-2.d0) + c(21)*rt**(-3.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**5.d0*(c(22)*rt**(-2.d0) + c(23)*rt**(-4.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**7.d0*(c(24)*rt**(-2.d0) + c(25)*rt**(-3.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**9.d0*(c(26)*rt**(-2.d0) + c(27)*rt**(-4.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**11.d0*(c(28)*rt**(-2.d0) + c(29)*rt**(-3.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**13.d0*(c(30)*rt**(-2.d0) + c(31)*rt**(-3.d0)
     &     + c(32)*rt**(-4.d0))
     &     *dexp(-gam*rr**2.d0)
c           put some units on pres (aJ/molecule)
      pres_pa = rpres * eps * xk / sig**3.d0 / 1.0e-30;
c           objective function
            fobj = pres_targ - pres_pa;
            return
            end
```

Appendix C. FORTRAN Code for Lennard-Jones EOS, mixture, given ρ, find P

```
      program lj_eos_mix
c     This program calculates the pressure using the equation of state
c     for the Lennard Jones fluid
c     Ref : Nicolas et. al. , "Molecular Physics, 1979, vol. 37, No. 5
c  pp 1429 - 1454.
c     Author : Parag Adhangale
c     Date created : March 10, 00.
c     Date last modified : Jan 26, 2002.
c     Modified by David Keffer
c     Date last modified : Feb 6, 2002.
      implicit double precision (a-h,o-z)
      integer, parameter :: ncomp = 2
      double precision, parameter :: T = 300.0d0 ! T (K)
      double precision, parameter :: vn = 275.0d0 ! molar volume (A^3/molecule)
      double precision, parameter :: xk = 1.38066e-23 ! Boltzmann's Constant
      double precision, dimension(1:32) :: c
      double precision, dimension(1:ncomp) :: sigvec, epsvec, xmol
      double precision, dimension(1:ncomp,1:ncomp) :: epsmat, kmat
      double precision :: k12
c
c     convert pure component parameters to mixture parameters
c
      epsvec(1) = 137.0d0       ! K
      epsvec(2) = 230.0d0       ! K
      sigvec(1) = 3.8820d0      ! Angstrom
      sigvec(2) = 4.4180d0      ! Angstrom
      xmol(1) = 0.50d0
      xmol(2) = 0.50d0
c     binary interaction parameter
c     need these for all i and j such that i<j
      k12 = 0.d0
c     check mole fractions
      xsum = sum(xmol)
      if (abs(xsum - 1.d0) .gt. 1.0d-12) then
c           if mole fractions don't sum to one, then normalize
            if (abs(xsum) .gt. 1.0d-12) then
                  xmol = xmol/xsum
            else
c           if mole fractions sum to zero, then make an equimolar mixture
                  xmol(1:ncomp) = 1.d0/dfloat(ncomp)
            endif
      endif
c     fill in diagonal of binary interaction matrix
      do i = 1, ncomp, 1
            kmat(i,i) = 0.d0
      enddo
c     fill in upper triangle of binary interaction matrix
      kmat(1,2) = k12;
c     use symmetry to fill in lower triangle of binary interaction matrix
      do j = 2, ncomp, 1
            do i = 1, j-1, 1
                  kmat(j,i) = kmat(i,j)
            enddo
```

```
            enddo
c           average sigma
            sig = 0.d0
            do i = 1, ncomp, 1
                  sig = sig + sigvec(i)*xmol(i)
            enddo
c           average epsilon
            do i = 1, ncomp, 1
                  do j = 1, ncomp, 1
                        epsmat(i,j) = dsqrt(epsvec(i)*epsvec(j))*(1.d0-kmat(i,j))
                  enddo
            enddo
            eps = 0.d0
            do i = 1, ncomp, 1
                  do j = 1, ncomp, 1
                        eps = eps + epsmat(i,j)*xmol(i)*xmol(j)
                  enddo
            enddo
            print *, 'sigma for mixture = ', sig, ' Angstroms'
            print *, 'epsilon for mixture = ', eps, ' K '
            rho = 1.d0/vn
c
c  Constants for the LJ eos :
c
      gam = 3.000000e+00
      c (1) = -0.44480725e-01
      c (2) = 00.72738221e+01
      c (3) = -0.14343368e+02
      c (4) = 00.38397096e+01
      c (5) = -0.20057745e+01
      c (6) = 00.19084472e+01
      c (7) = -0.57441787e+01
      c (8) = 00.25110073e+02
      c (9) = -0.45232787e+04
      c (10)= 00.89327162e-02
      c (11)= 00.98163358e+01
      c (12)= -0.61434572e+02
      c (13)= 00.14161454e+02
      c (14)= 00.43353841e+02
      c (15)= 00.11078327e+04
      c (16)= -0.35429519e+02
      c (17)= 00.10591298e+02
      c (18)= 00.49770046e+03
      c (19)= -0.35338542e+03
      c (20)= 00.45036093e+04
      c (21)= 00.77805296e+01
      c (22)= 00.13567114e+05
      c (23)= -0.85818023e+01
      c (24)= 00.16646578e+05
      c (25)= -0.14092234e+02
      c (26)= 00.19386911e+05
      c (27)= 00.38585868e+02
      c (28)= 00.33800371e+04
      c (29)= -0.18567754e+03
      c (30)= 00.84874693e+04
      c (31)= 00.97508689e+02
```

```
      c (32)= -0.14483060e+02
c           calculate reduced T, rho, and pres
      rt = T / eps
      rr = rho * sig**3.d0;
      rpres = rr * rt + rr**2.d0 * (c(1) * rt + c(2)*rt**(1.d0/2.d0)
     &     + c(3) +  c(4)*rt**(-1.d0) + c(5)*rt**(-2.d0))
     &     +  rr**3.d0*(c(6)*rt + c(7) + c(8)*rt**(-1.d0)
     &     + c(9)*rt**(-2.d0))
     &     +  rr**4.d0*(c(10)*rt + c(11) + c(12)*rt**(-1.d0))
     &     + rr**5.d0*c(13)
     &     +  rr**6.d0*(c(14)*rt**(-1.d0) + c(15)*rt**(-2.d0))
     &     +  rr**7.d0*(c(16)*rt**(-1.d0)) + rr**8.d0*(c(17)*rt**(-1.d0)
     &     +  c(18)*rt**(-2.d0)) + rr**9.d0*(c(19)*rt**(-2.d0))
     &     +  rr**3.d0*(c(20)*rt**(-2.d0) + c(21)*rt**(-3.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**5.d0*(c(22)*rt**(-2.d0) + c(23)*rt**(-4.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**7.d0*(c(24)*rt**(-2.d0) + c(25)*rt**(-3.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**9.d0*(c(26)*rt**(-2.d0) + c(27)*rt**(-4.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**11.d0*(c(28)*rt**(-2.d0) + c(29)*rt**(-3.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**13.d0*(c(30)*rt**(-2.d0) + c(31)*rt**(-3.d0)
     &     + c(32)*rt**(-4.d0))
     &     *dexp(-gam*rr**2.d0)
c           put some units on pres (aJ/molecule)
      pres_pa = rpres * eps * xk / sig**3.d0 / 1.0e-30;
c           convert from Pa to bar
            pres_bar = pres_pa*1.0d-5;
c           print *, ' sigma = ', sig, ' Angstroms & epsilon = ', eps, ' K '
            print *, ' T = ', T, ' K & Vn = ', vn, ' Angstroms^3/molecule '
            print *, 'Pressure = ', pres_pa, ' Pa = ', pres_bar, ' bar '
            stop
            end
```

Appendix D. FORTRAN Code for Lennard-Jones EOS, mixture, given P, find ρ

```
      program lj_eos_mix_TP
c     This program uses a Newton Raphson method
c     with numerical derivatives to solve for
c     the molar volume, given temperature and pressure,
c     for the Lennard Jones Equation of State
c     with mixture rules
c
c     Author:  David Keffer
c     Begun:   02/05/02
c     completed:  02/05/02
c
c     Modified by David Keffer
c     Date last modified : Feb 6, 2002.
      implicit double precision (a-h,o-z)
      integer, parameter :: ncomp = 2
      double precision, parameter :: T = 300.0d0 ! T (K)
      double precision, parameter :: pres_targ = 101325.0d0 ! Pressure (Pa)
      double precision, dimension(1:ncomp) :: sigvec, epsvec, xmol
      double precision, dimension(1:ncomp,1:ncomp) :: epsmat, kmat
      double precision :: k12
c
c     convert pure component parameters to mixture parameters
c
      epsvec(1) = 137.0d0       ! K
      epsvec(2) = 230.0d0       ! K
      sigvec(1) = 3.8820d0      ! Angstrom
      sigvec(2) = 4.4180d0      ! Angstrom
      xmol(1) = 0.50d0
      xmol(2) = 0.50d0
c     binary interaction parameter
c     need these for all i and j such that i<j
      k12 = 0.d0
c     check mole fractions
      xsum = sum(xmol)
      if (abs(xsum - 1.d0) .gt. 1.0d-12) then
c           if mole fractions don't sum to one, then normalize
            if (abs(xsum) .gt. 1.0d-12) then
                  xmol = xmol/xsum
            else
c           if mole fractions sum to zero, then make an equimolar mixture
                  xmol(1:ncomp) = 1.d0/dfloat(ncomp)
            endif
      endif
c     fill in diagonal of binary interaction matrix
      do i = 1, ncomp, 1
            kmat(i,i) = 0.d0
      enddo
c     fill in upper triangle of binary interaction matrix
      kmat(1,2) = k12;
c     use symmetry to fill in lower triangle of binary interaction matrix
      do j = 2, ncomp, 1
            do i = 1, j-1, 1
                  kmat(j,i) = kmat(i,j)
```

```
                enddo
        enddo
c       average sigma
        sig = 0.d0
        do i = 1, ncomp, 1
                sig = sig + sigvec(i)*xmol(i)
        enddo
c       average epsilon
        do i = 1, ncomp, 1
                do j = 1, ncomp, 1
                        epsmat(i,j) = dsqrt(epsvec(i)*epsvec(j))*(1.d0-kmat(i,j))
                enddo
        enddo
        eps = 0.d0
        do i = 1, ncomp, 1
                do j = 1, ncomp, 1
                        eps = eps + epsmat(i,j)*xmol(i)*xmol(j)
                enddo
        enddo
        print *, 'sigma for mixture = ', sig, ' Angstroms'
        print *, 'epsilon for mixture = ', eps, ' K '
c
c       set the initial guess for the molar volume (A^3/molecule)
        vn_old = 275.d0
c
c Newton-Raphson method with numerical approximations to the derivative.
c

        maxit = 100;
        tol = 1.0d-8;
        err = 100.d0;
        icount = 0;
        xold = vn_old;
        do while (err .gt. tol .and. icount .le. maxit)
                icount = icount + 1;
                call funkeval(sig, eps, T, xold, f, pres_targ)
                print *, ' i= ', icount, ' vn= ', xold, ' f= ', f
                h = min(0.01*xold,0.01)
                call dfunkeval(xold, h, sig, eps, T, df, pres_targ)
                xnew = xold - f/df
                if (icount .gt. 1) then
                        err = abs((xnew - xold)/xnew)
                endif
                xold = xnew
                print *, ' i= ', icount, ' vn= ', xold, ' err= ', err
        enddo
        x0 = xnew
        if (icount .ge. maxit) then
c               you ran out of iterations
                print *, 'Sorry. You did not converge in ',maxit,' iterations'
                print *, 'The final value of x was ', x0
        endif
        vn = x0
        pres_pa = pres_targ
        pres_bar = pres_pa*1.0d-5;
        print *, ' sigma = ', sig, ' Angstroms & epsilon = ', eps, ' K '
```

```
      print *, ' T = ', T, ' K & Vn = ', vn, ' Angstroms^3/molecule '
      print *, 'Pressure = ', pres_pa, ' Pa = ', pres_bar, ' bar '
      stop
      end

c
c     This subroutine calculates a numerical approximation to the derivative
c
      subroutine dfunkeval(vn, h, sig, eps, T, df, pres_targ)
      implicit double precision (a-h,o-z)
      double precision, intent(in) :: sig, eps, T, vn, h, pres_targ
      double precision, intent(out) :: df
      call funkeval(sig, eps, T, vn+h, fp, pres_targ)
      call funkeval(sig, eps, T, vn-h, fn, pres_targ)
      df = (fp - fn)/(2*h);
      return
      end

c
c     This subroutine evaluates the function
c
      subroutine funkeval(sig, eps, T, vn, fobj, pres_targ)
c     This program calculates the pressure using the equation of state
c     for the Lennard Jones fluid
c     Ref : Nicolas et. al. , "Molecular Physics, 1979, vol. 37, No. 5
c    pp 1429 - 1454.
c     Author : Parag Adhangale
c     Date created : March 10, 00.
c     Date last modified : Jan 26, 2002.
      implicit double precision (a-h,o-z)
      double precision, intent(in) :: sig, eps, T, vn, pres_targ
      double precision, intent(out) :: fobj
      double precision, parameter :: xk = 1.38066e-23 ! Boltzmann's Constant
      double precision, dimension(1:32) :: c
      rho = 1.d0/vn
c
c  Constants for the LJ eos :
c
    gam = 3.000000e+00
    c (1) = -0.44480725e-01
    c (2) = 00.72738221e+01
    c (3) = -0.14343368e+02
    c (4) = 00.38397096e+01
    c (5) = -0.20057745e+01
    c (6) = 00.19084472e+01
    c (7) = -0.57441787e+01
    c (8) = 00.25110073e+02
    c (9) = -0.45232787e+04
    c (10)= 00.89327162e-02
    c (11)= 00.98163358e+01
    c (12)= -0.61434572e+02
    c (13)= 00.14161454e+02
    c (14)= 00.43353841e+02
    c (15)= 00.11078327e+04
    c (16)= -0.35429519e+02
    c (17)= 00.10591298e+02
```

```
      c (18)= 00.49770046e+03
      c (19)= -0.35338542e+03
      c (20)= 00.45036093e+04
      c (21)= 00.77805296e+01
      c (22)= 00.13567114e+05
      c (23)= -0.85818023e+01
      c (24)= 00.16646578e+05
      c (25)= -0.14092234e+02
      c (26)= 00.19386911e+05
      c (27)= 00.38585868e+02
      c (28)= 00.33800371e+04
      c (29)= -0.18567754e+03
      c (30)= 00.84874693e+04
      c (31)= 00.97508689e+02
      c (32)= -0.14483060e+02
c           calculate reduced T, rho, and pres
      rt = T / eps
      rr = rho * sig**3.d0;
      rpres = rr * rt + rr**2.d0 * (c(1) * rt + c(2)*rt**(1.d0/2.d0)
     &     + c(3) +  c(4)*rt**(-1.d0) + c(5)*rt**(-2.d0))
     &     +  rr**3.d0*(c(6)*rt + c(7) + c(8)*rt**(-1.d0)
     &     + c(9)*rt**(-2.d0))
     &     +  rr**4.d0*(c(10)*rt + c(11) + c(12)*rt**(-1.d0))
     &     + rr**5.d0*c(13)
     &     +  rr**6.d0*(c(14)*rt**(-1.d0) + c(15)*rt**(-2.d0))
     &     +  rr**7.d0*(c(16)*rt**(-1.d0)) + rr**8.d0*(c(17)*rt**(-1.d0)
     &     +  c(18)*rt**(-2.d0)) + rr**9.d0*(c(19)*rt**(-2.d0))
     &     +  rr**3.d0*(c(20)*rt**(-2.d0) + c(21)*rt**(-3.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**5.d0*(c(22)*rt**(-2.d0) + c(23)*rt**(-4.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**7.d0*(c(24)*rt**(-2.d0) + c(25)*rt**(-3.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**9.d0*(c(26)*rt**(-2.d0) + c(27)*rt**(-4.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**11.d0*(c(28)*rt**(-2.d0) + c(29)*rt**(-3.d0))
     &     *dexp(-gam*rr**2.d0)
     &     +  rr**13.d0*(c(30)*rt**(-2.d0) + c(31)*rt**(-3.d0)
     &     + c(32)*rt**(-4.d0))
     &     *dexp(-gam*rr**2.d0)
c           put some units on pres (aJ/molecule)
      pres_pa = rpres * eps * xk / sig**3.d0 / 1.0e-30;
c           objective function
            fobj = pres_targ - pres_pa;
            return
            end
```