```
            // relable variables for next iteration
            // x1new = x2old, y1new=y2old
            x1 = x2 ;
            y1 = y2 ;

            // next iteration values for x2, y2
            x2 = x2 + delx ;
            pdetail(x2) ;
            y2 = dPCalc - dP ;
        }

        // ddetail: maximum number of iterations exceeded
        ptAGA10->lStatus=MAX_NUM_OF_ITERATIONS_EXCEEDED ;
        dRho = x2 ;
}       // Detail::ddetail()
```

```
/*************************************************************************
 *    Function    :     Detail::braket()
 *    Arguments   :     AGA10STRUCT *
 *    Returns     :     void
 *    Purpose     :     brackets density solution
 *    Revisions   :
 *************************************************************************/

// Note: this function was ported from the AGA Report No.8 FORTRAN listing,
//           retaining as much of the original content as possible

void Detail::braket(AGA10STRUCT *ptAGA10)
{
      // variables local to function
      int imax, it ;
      double del, rhomax, videal ;
      double rho1, rho2, p1, p2 ;

      // initialize
      imax = 200 ;
      rho1 = 0.0 ;
      p1 = 0.0 ;
      rhomax = 1.0 / dKp3 ;
      if (dT > 1.2593 * dU) rhomax = 20.0 * rhomax ;
      videal = RGASKJ * dT / dP ;

      if (fabs(dB) < (0.167 * videal))
      {
            rho2 = 0.95 / (videal + dB) ;
      }
      else
      {
            rho2 = 1.15 / videal ;
      }

      del = rho2 / 20.0 ;

      // start iterative density search loop
      for (it = 0; it < imax ; it++)
      {
            if (rho2 > rhomax && ptAGA10->lStatus != MAX_DENSITY_IN_BRAKET_EXCEEDED)
```

```
{
      // density in braket exceeds maximum allowable density
      ptAGA10->lStatus = MAX_DENSITY_IN_BRAKET_EXCEEDED ;
      del = 0.01 * (rhomax - rho1) + (dP / (RGASKJ * dT)) / 20.0 ;
      rho2 = rho1 + del ;
      continue ;
}

// calculate pressure p2 at density rho2
pdetail(rho2) ;
p2 = dPCalc ;

// test value of p2 relative to p and relative to p1
if (p2 > dP)
{
      // the density root is bracketed (p1<p and p2>p)
      dRhoL = rho1 ;
      dPRhoL = p1 ;
      dRhoH = rho2 ;
      dPRhoH = p2 ;
      ptAGA10->lStatus = NORMAL ;
      return;
}

else if (p2 > p1)
{
      if (ptAGA10->lStatus == MAX_DENSITY_IN_BRAKET_EXCEEDED) del *= 2.0 ;
      rho1 = rho2 ;
      p1 = p2 ;
      rho2 = rho1 + del ;
      continue ;
}
else
{
      // lStatus= NEGATIVE_DENSITY_DERIVATIVEindicates that
      // pressure has a negative density derivative, since p2 is less than
      // some previous pressure
      ptAGA10->lStatus = NEGATIVE_DENSITY_DERIVATIVE;
      dRho = rho1;
      return;
}
```

```
        }
        // maximum number of iterations exceeded if we fall through the bottom
        ptAGA10->lStatus = MAX_NUM_OF_ITERATIONS_EXCEEDED ;
        dRho = rho2 ;
        return ;

}       // Detail::braket()


/**********************************************************************
 *      Function    :       Detail::pdetail()
 *      Arguments   :       double
 *      Returns     :       void
 *      Purpose     :       calculates pressure, given D and T. Calls zdetail()
 *      Revisions   :
 **********************************************************************/

void Detail::pdetail(double dD)
{
        dPCalc = zdetail(dD) * dD * RGASKJ * dT ;
}       // Detail::pdetail()
```

```
/**************************************************************************
 *    Function    :     Detail::zdetail()
 *    Arguments   :     double
 *    Returns     :     void
 *    Purpose     :     calculates compressibility
 *    Revisions   :
 **************************************************************************/

double Detail::zdetail(double d)
{
      // variables local to function
      double D1, D2, D3, D4, D5, D6, D7, D8, D9, exp1, exp2, exp3, exp4 ;

      // powers of reduced density
      D1 = dKp3 * d ;
      D2 = D1 * D1 ;
      D3 = D2 * D1 ;
      D4 = D3 * D1 ;
      D5 = D4 * D1 ;
      D6 = D5 * D1 ;
      D7 = D6 * D1 ;
      D8 = D7 * D1 ;
      D9 = D8 * D1 ;

      exp1 = exp(-D1) ;
      exp2 = exp(-D2) ;
      exp3 = exp(-D3) ;
      exp4 = exp(-D4) ;

      // the following expression for Z was adopted from FORTRAN example in AGA8
      dZ = 1.0 + dB * d
            +  adFn[12] * D1 * (exp3 - 1.0 - 3.0 * D3 * exp3)
            + (adFn[13] + adFn[14] + adFn[15]) * D1 * (exp2 - 1.0 - 2.0 * D2 * exp2)
            + (adFn[16] + adFn[17]) * D1 * (exp4 - 1.0 - 4.0 * D4 * exp4)
            + (adFn[18] + adFn[19]) * D2 * 2.0
            + (adFn[20] + adFn[21] + adFn[22]) * D2 * (2.0 - 2.0 * D2) * exp2
            + (adFn[23] + adFn[24] + adFn[25]) * D2 * (2.0 - 4.0 * D4) * exp4
            +  adFn[26] * D2 * (2.0 - 4.0 * D4) * exp4
            +  adFn[27] * D3 * 3.0
            + (adFn[28] + adFn[29]) * D3 * (3.0 - D1) * exp1
            + (adFn[30] + adFn[31]) * D3 * (3.0 - 2.0 * D2) * exp2
```

```
            + (adFn[32] + adFn[33]) * D3 * (3.0 - 3.0 * D3) * exp3
            + (adFn[34] + adFn[35] + adFn[36]) * D3 * (3.0 - 4.0 * D4) * exp4
            + (adFn[37] + adFn[38]) * D4 * 4.0
            + (adFn[39] + adFn[40] + adFn[41]) * D4 * (4.0 - 2.0 * D2) * exp2
            + (adFn[42] + adFn[43]) * D4 * (4.0 - 4.0 * D4) * exp4
            +  adFn[44] * D5 * 5.0
            + (adFn[45] + adFn[46]) * D5 * (5.0 - 2.0 * D2) * exp2
            + (adFn[47] + adFn[48]) * D5 * (5.0 - 4.0 * D4) * exp4
            +  adFn[49] * D6 * 6.0
            +  adFn[50] * D6 * (6.0 - 2.0 * D2) * exp2
            +  adFn[51] * D7 * 7.0
            +  adFn[52] * D7 * (7.0 - 2.0 * D2) * exp2
            +  adFn[53] * D8 * (8.0 - D1) * exp1
            + (adFn[54] + adFn[55]) * D8 * (8.0 - 2.0 * D2) * exp2
            + (adFn[56] + adFn[57]) * D9 * (9.0 - 2.0 * D2) * exp2 ;

      return dZ ;
}     // Detail::zdetail()



/*************************************************************************
 *    Function    :     Detail::dZdT()
 *    Arguments   :     double
 *    Returns     :     double
 *    Purpose     :     calculates the first partial derivative of Z wrt T
 *    Revisions   :
 *************************************************************************/

double Detail::dZdT(double d)
{
      // variables local to function
      double tmp ;
      int i ;
      double D1, D2, D3, D4, D5, D6, D7, D8, D9, exp1, exp2, exp3, exp4 ;

      // set up powers of reduced density
      D1 = dKp3 * d ;
      D2 = D1 * D1 ;
      D3 = D2 * D1 ;
      D4 = D3 * D1 ;
```

```
D5 = D4 * D1 ;
D6 = D5 * D1 ;
D7 = D6 * D1 ;
D8 = D7 * D1 ;
D9 = D8 * D1 ;

exp1 = exp(-D1) ;
exp2 = exp(-D2) ;
exp3 = exp(-D3) ;
exp4 = exp(-D4) ;

// create terms uC*T^-(un+1) from coefficients we've already computed (An[n])
for (i=12; i < 58; i++)
{
      if (adUn[i] && adFn[i])
      {
            fx[i] = (adFn[i] * adUn[i] * D1) / dT;
      }
      else
      {
            fx[i] = 0.0 ;
      }
}

// initial part of equation
ddZdT = d * ddBdT ;

// n=13 evaluates to zero except for hydrogen, for whom fn = 1
if (dF) ddZdT += fx[12] - (fx[12] * (1.0 - 3.0 * D3) * exp3) ;

tmp = (1.0 - 2.0 * D2) * exp2 ;
ddZdT += (fx[13] - (fx[13] * tmp)) ;
ddZdT += fx[14] - (fx[14] * tmp) ;
ddZdT += fx[15] - (fx[15] * tmp) ;

tmp = (1.0 - 4.0 * D4) * exp4 ;
ddZdT += fx[16] - (fx[16] * tmp) ;
ddZdT += fx[17] - (fx[17] * tmp) ;

ddZdT = ddZdT - (fx[18] + fx[19]) * D1 * 2.0
      - (fx[21] + fx[22]) * D1 * (2.0 - 2.0 * D2) * exp2
```

```
            - (fx[23] + fx[24] + fx[25]) * D1 * (2.0 - 4.0 * D4) * exp4
            - fx[26] * D1 * (2.0 - 4.0 * D4) * exp4
            - fx[27] * D2 * 3.0
            - (fx[28] + fx[29]) * D2 * (3.0 - D1) * exp1
            - (fx[30] + fx[31]) * D2 * (3.0 - 2.0 * D2) * exp2
            - (fx[32] + fx[33]) * D2 * (3.0 - 3.0 * D3) * exp3
            - (fx[34] + fx[35] + fx[36]) * D2 * (3.0 - 4.0 * D4) * exp4
            - fx[37] * D3 * 4.0
            - (fx[39] + fx[40] + fx[41]) * D3 * (4.0 - 2.0 * D2) * exp2
            - (fx[42] + fx[43]) * D3 * (4.0 - 4.0 * D4) * exp4
            - fx[44] * D4 * 5.0
            - (fx[45] + fx[46]) * D4 * (5.0 - 2.0 * D2) * exp2
            - (fx[47] + fx[48]) * D4 * (5.0 - 4.0 * D4) * exp4
            - fx[49] * D5 * 6.0
            - fx[50] * D5 * (6.0 - 2.0 * D2) * exp2
            - fx[51] * D6 * 7.0
            - fx[52] * D6 * (7.0 - 2.0 * D2) * exp2
            - fx[53] * D7 * (8.0 - D1) * exp1
            - fx[54] * D7 * (8.0 - 2.0 * D2) * exp2
            - fx[56] * D8 * (9.0 - 2.0 * D2) * exp2 ;

    return ddZdT ;
}   // Detail::dDdT()
```

```
/*************************************************************************
 *    Function    :     Detail::d2ZdT2()
 *    Arguments   :     double
 *    Returns     :     double
 *    Purpose     :     calculates the second partial derivative of Z wrt T
 *    Revisions   :
 *************************************************************************/

double Detail::d2ZdT2(double d)
{
      // variables local to function
      double tmp ;
      int i ;
      double D1, D2, D3, D4, D5, D6, D7, D8, D9, exp1, exp2, exp3, exp4 ;

      // set up powers of reduced density
      D1 = dKp3 * d ;
      D2 = D1 * D1 ;
      D3 = D2 * D1 ;
      D4 = D3 * D1 ;
      D5 = D4 * D1 ;
      D6 = D5 * D1 ;
      D7 = D6 * D1 ;
      D8 = D7 * D1 ;
      D9 = D8 * D1 ;
      exp1 = exp(-D1) ;
      exp2 = exp(-D2) ;
      exp3 = exp(-D3) ;
      exp4 = exp(-D4) ;

      // create terms uC*T^-(un+1) from coefficients we've already computed (An[n])
      for (i=12; i < 58; i++)
      {
            if (adUn[i] && adFn[i])
            {
                  fx[i] = (adFn[i] * D1 * adUn[i] * (adUn[i] + 1.0)) / (dT * dT) ;
            }
            else
            {
                  fx[i] = 0.0 ;
            }
```

```
}

// initial part of equation
dd2ZdT2 = d * dd2BdT2 ;

// n=13 evaluates to zero except for hydrogen, for whom fn = 1
if (dF) dd2ZdT2 += fx[12] - (fx[12] * (1.0 - 3.0 * D3) * exp3) ;

tmp = (1.0 - 2.0 * D2) * exp2 ;
dd2ZdT2 += -fx[13] + (fx[13] * tmp) ;
dd2ZdT2 += -fx[14] + (fx[14] * tmp) ;
dd2ZdT2 += -fx[15] + (fx[15] * tmp) ;

tmp = (1.0 - 4.0 * D4) * exp4 ;
dd2ZdT2 += -fx[16] + (fx[16] * tmp) ;
dd2ZdT2 += -fx[17] + (fx[17] * tmp) ;

dd2ZdT2 = dd2ZdT2 + (fx[18] + fx[19]) * D1 * 2.0
      + (fx[21] + fx[22]) * D1 * (2.0 - 2.0 * D2) * exp2
      + (fx[23] + fx[24] + fx[25]) * D1 * (2.0 - 4.0 * D4) * exp4
      + fx[26] * D1 * (2.0 - 4.0 * D4) * exp4
      + fx[27] * D2 * 3.0
      + (fx[28] + fx[29]) * D2 * (3.0 - D1) * exp1
      + (fx[30] + fx[31]) * D2 * (3.0 - 2.0 * D2) * exp2
      + (fx[32] + fx[33]) * D2 * (3.0 - 3.0 * D3) * exp3
      + (fx[34] + fx[35] + fx[36]) * D2 * (3.0 - 4.0 * D4) * exp4
      + fx[37] * D3 * 4.0
      + (fx[39] + fx[40] + fx[41]) * D3 * (4.0 - 2.0 * D2) * exp2
      + (fx[42] + fx[43]) * D3 * (4.0 - 4.0 * D4) * exp4
      + fx[44] * D4 * 5.0
      + (fx[45] + fx[46]) * D4 * (5.0 - 2.0 * D2) * exp2
      + (fx[47] + fx[48]) * D4 * (5.0 - 4.0 * D4) * exp4
      + fx[49] * D5 * 6.0
      + fx[50] * D5 * (6.0 - 2.0 * D2) * exp2
      + fx[51] * D6 * 7.0
      + fx[52] * D6 * (7.0 - 2.0 * D2) * exp2
      + fx[53] * D7 * (8.0 - D1) * exp1
      + fx[54] * D7 * (8.0 - 2.0 * D2) * exp2
      + fx[56] * D8 * (9.0 - 2.0 * D2) * exp2 ;

return dd2ZdT2 ;
```

```
}     // Detail::d2ZdT2()


/**********************************************************************
 *    Function    :     Detail::dZdD()
 *    Arguments   :     double
 *    Returns     :     double
 *    Purpose     :     calculates the first partial derivative of Z wrt D
 *    Revisions   :
 **********************************************************************/

// For efficiency and continuity with AGA 8 code example, each term
// is evaluated individually rather than through looping through tables.
// Temporary storage is used to hold portions of complex equations and
// to facilitate debugging. Additional speed optimization is possible.

double Detail::dZdD(double d)
{
      double temp, temp1, temp2, temp3;
      int i ;
      double D1, D2, D3, D4, D5, D6, D7, D8, D9, exp1, exp2, exp3, exp4 ;

      // set up powers of reduced density
      D1 = dKp3 * d ;
      D2 = D1 * D1 ;
      D3 = D2 * D1 ;
      D4 = D3 * D1 ;
      D5 = D4 * D1 ;
      D6 = D5 * D1 ;
      D7 = D6 * D1 ;
      D8 = D7 * D1 ;
      D9 = D8 * D1 ;
      exp1 = exp(-D1) ;
      exp2 = exp(-D2) ;
      exp3 = exp(-D3) ;
      exp4 = exp(-D4) ;

      // create terms uC*T^-(un+1) from coefficients we've already computed (An[n])
      for (i=12; i < 58; i++)
      {
```