554 lines

```
PROGRAM oned_diff
IMPLICIT NONE

! Declaration module
REAL :: t, p, s, c_b, d, d_c, m_d, m_b, m_j, m_k, r_ave, lam, z_b, n_x,
n_y
REAL :: t_c, n_z, s_x, s_y, s_z, beta_b, beta_d
DOUBLE PRECISION :: pi, del_z, del_zd, x_run, r_run, fr, sum_a, speed,
adj
DOUBLE PRECISION :: energy, r_b, r_d, v_b, v_d, x(95000), y(95000), z
(95000)
DOUBLE PRECISION :: xc(95000), yc(95000), zc(95000), vxc(95000), vyc
(95000)
DOUBLE PRECISION :: vzc(95000), vx(95000), vy(95000), vz(95000), x_j
(95000)
DOUBLE PRECISION :: y_j(95000), z_j(95000), x_k(95000), y_k(95000), z_k
(95000)
DOUBLE PRECISION :: del_v, time, delta_t, dx, dy, dz, r_check, dist, phi
INTEGER :: n_d, i, j, k, q, sum_bi, it, itm, n_bd, coll_2b, coll_bd,
n_ref
INTEGER :: n_b, hit(95000), n_cd(1000), n_steps, steps, coll_2d, o_o, o
INTEGER :: tag(95000), n_cb(1000), n_sb, n_zb, n_sd, sum_ter, n_pair,
k_b, o_b
CHARACTER (LEN = 1) :: answer

! Input module
PRINT *, "Input the temperature in Kelvin"
READ *, t
PRINT *, "Input the pressure in atm for the bath molecules"
READ *, p
del_z = (.13626*t/p)**(1./3.) ! Cell size
PRINT *, "the cell size in nm is", del_z
PRINT *, "Input the number of bath molecules along one of the non-
diffusing &
(x and y)"
PRINT *, "coordinates"
READ *, n_sb
s = del_z*float(n_sb)
PRINT *, "The xy area of the container is", s**2., "nm^2"
PRINT *, "Input the number of bath molecules along the diffusing (z)
coordinate"
READ *, n_zb
d = del_z*n_zb
PRINT *, "The bin length is", d, "nm"
n_b = n_sb*n_sb*n_zb
PRINT *, "The number of bath molecules in each bin is", n_b
PRINT *, "Input the molar mass of the bath molecules in kg/mole"
READ *, m_b
PRINT *, "Input the molar mass of the diffusing molecules"
READ *, m_d
PRINT *, "Input the molecular radius of the bath molecules in nm"
READ *, r_b
PRINT *, "Input the molecular radius of the diffusing molecules"
READ *, r_d
v_b = (4.994347E9)*sqrt(t/m_b) ! Root-mean square speeds in nm/sec
v_d = (4.994347E9)*sqrt(t/m_d)
beta_b = (6.01361E-20)*m_b/t
beta_d = (6.01361E-20)*m_d/t
pi = 3.141592653589
lam = .043373*t/(p*sqrt(1.+m_d/m_b)*(r_b+r_d)**2.)
PRINT *, "The mean free path for the dilute diffusing molecules is", lam
```

```
PRINT *, "The collision probability within a bin is", 1.-exp(-d/(lam*cos
(1.)))
DO k = 1, 100
  PRINT *, "Input the number of diffusing molecules along the x
coordinate"
  READ *, n_sd
  IF (n_sd.lt.(s/(2.5*r_d))) THEN
    EXIT
  ENDIF
  PRINT *, "The number of diffusing molecules is too large. Try another
number"
ENDDO
del_zd = s/float(n_sd)
n_d = n_sd**2.
PRINT *, "The number of diffusing molecules is", n_d
PRINT *, "They are spaced", del_zd, "nm apart in the xy plane"
! Coordinate placement
q = 1
k = 1
DO i = 1, n_sd
  DO j = 1, n_sd
    x(q) = (float(i)-.5)*del_zd+(.125*r_d/d)*cos(sqrt(float(j+q)))
    y(q) = (float(j)-.5)*del_zd+(.125*r_d/d)*sin(sqrt(float(j+i)))
    z(q) = (.5-float(k))*del_z+(.25*r_d/d)*sin(sqrt(float(i+q)))
    IF (k.eq.5) THEN
      k = 0
    ENDIF
    k = k+1
    q = q+1
  ENDDO
ENDDO
q = 1+n_d
DO i = 1, n_sb
  DO j = 1, n_sb
    DO k = 1, n_zb
      x(q) = (float(i)-.5)*del_z+(.25*r_b/d)*sin(sqrt(float(j+q)))
      y(q) = (float(j)-.5)*del_z+(.25*r_b/d)*cos(sqrt(float(k+q)))
      z(q) = (float(k)-.5)*del_z+(.25*r_b/d)*cos(sqrt(float(i+k)))
      q = q+1
    ENDDO
  ENDDO
ENDDO

! Velocity vectors
n_x = 0.1870133
n_y = 0.2093135
n_z = 0.3698417
s_x = -1.
s_y = 1.
s_z = 1.
energy = 0.
del_v = .00005/sqrt(beta_d)
DO i = 1, n_d
  n_x = n_x+.2730911
  IF (n_x.ge.1.) THEN
    n_x = n_x-1.
  ENDIF
  n_y = n_y+.2300787
  IF (n_y.ge.1.) THEN
    n_y = n_y-1.
  ENDIF
```

```
    n_z = n_z+.2911071
    IF (n_z.ge.1.) THEN
      n_z = n_z-1.
    ENDIF
    vx(i) = .5*del_v
    vy(i) = .5*del_v
    vz(i) = .5*del_v
    sum_a = 0.
    DO j = 1, 500000
      sum_a = sum_a+2.*sqrt(beta_d/pi)*del_v*exp(-beta_d*vx(i)*vx(i))
      IF (sum_a.ge.n_x) THEN
        EXIT
      ENDIF
      vx(i) = vx(i)+del_v
      s_x = -s_x
    ENDDO
    sum_a = 0.
    DO j = 1, 500000
      sum_a = sum_a+2.*sqrt(beta_d/pi)*del_v*exp(-beta_d*vy(i)*vy(i))
      IF (sum_a.ge.n_y) THEN
        EXIT
      ENDIF
      vy(i) = vy(i)+del_v
      s_y = -s_y
    ENDDO
    sum_a = 0.
    DO j = 1, 500000
      sum_a = sum_a+2.*sqrt(beta_d/pi)*del_v*exp(-beta_d*vz(i)*vz(i))
      IF (sum_a.ge.n_z) THEN
        EXIT
      ENDIF
      vz(i) = vz(i)+del_v
      s_z = -s_z
    ENDDO
    vx(i) = s_x*vx(i)
    vy(i) = s_y*vy(i)
    vz(i) = s_z*vz(i)
    energy = energy+vx(i)**2.+vy(i)**2.+vz(i)**2.
  ENDDO
  t_c = m_d*energy/((2.49434E19)*float(n_d))
  adj = t/t_c
  energy = 0.
  del_v = .00005/sqrt(beta_b)
  DO i = 1, n_d
    vx(i) = sqrt(adj)*vx(i)
    vy(i) = sqrt(adj)*vy(i)
    vz(i) = sqrt(adj)*vz(i)
  ENDDO
  DO i = n_d+1, n_d+n_b
    n_x = n_x+.2199431
    IF (n_x.ge.1.) THEN
      n_x = n_x-1.
    ENDIF
    n_y = n_y+.3170933
    IF (n_y.ge.1.) THEN
      n_y = n_y-1.
    ENDIF
    n_z = n_z+.2609717
    IF (n_z.ge.1.) THEN
      n_z = n_z-1.
    ENDIF
```

```
  vx(i) = .5*del_v
  vy(i) = .5*del_v
  vz(i) = .5*del_v
  DO j = 1, 500000
    sum_a = sum_a+2.*sqrt(beta_b/pi)*del_v*exp(-beta_b*vx(i)*vx(i))
    IF (sum_a.ge.n_x) THEN
      EXIT
    ENDIF
    vx(i) = vx(i)+del_v
    s_x = -s_x
  ENDDO
  sum_a = 0.
  DO j = 1, 500000
    sum_a = sum_a+2.*sqrt(beta_b/pi)*del_v*exp(-beta_b*vy(i)*vy(i))
    IF (sum_a.ge.n_y) THEN
      EXIT
    ENDIF
    vy(i) = vy(i)+del_v
    s_y = -s_y
  ENDDO
  sum_a = 0.
  DO j = 1, 500000
    sum_a = sum_a+2.*sqrt(beta_b/pi)*del_v*exp(-beta_b*vz(i)*vz(i))
    IF (sum_a.ge.n_z) THEN
      EXIT
    ENDIF
    vz(i) = vz(i)+del_v
    s_z = -s_z
  ENDDO
  vx(i) = s_x*vx(i)
  vy(i) = s_y*vy(i)
  vz(i) = s_z*vz(i)
  energy = energy+vx(i)**2.+vy(i)**2.+vz(i)**2.
ENDDO
t_c = m_b*energy/((2.49434E19)*float(n_b))
adj = t/t_c
DO i = n_d+1, n_d+n_b
  vx(i) = sqrt(adj)*vx(i)
  vy(i) = sqrt(adj)*vy(i)
  vz(i) = sqrt(adj)*vz(i)
ENDDO
PRINT *, "Input the average distance traveled by a bath molecule in nm
per &
time step"
READ *, r_ave
delta_t = r_ave/v_b
PRINT *, "The time step in seconds is", delta_t
z_b = 60.08*r_b*r_b*float(n_b)*v_b*p/t
PRINT *, "The b-b collision frequency in sec^-1 is", z_b
PRINT *, "Input the number of time steps before diffusion is turned on"
READ *, n_steps
PRINT *, "Input the number of time steps before outputting data"
READ *, steps
it = 1
itm = 1
k_b = 1
coll_2b = 0
coll_bd = 0
coll_2d = 0
n_ref = 0
DO i = 1, 95000
```

```
  hit(i) = 0
  tag(i) = 0
ENDDO
sum_bi = 0
sum_ter = 0
time = 0.
x_run = 0.
r_run = 0.
fr = 0.
PRINT *, "  "
PRINT *, "The trajectory simulation now begins..............thank God!"

! Motion and output modules
DO o_o = 1, 20
  DO o = 1, steps
    IF (it.eq.n_steps) THEN
      q = 1
      DO i = 1, n_sd
        DO j = 1, n_sd
          x(q) = (float(i)-.5)*del_zd
          y(q) = (float(j)-.5)*del_zd
          z(q) = r_d
          vz(q) = abs(vz(q))
          q = q+1
        ENDDO
      ENDDO
    ENDIF
    ! Translation equations
    DO i = 1, n_d+k_b*n_b
      x(i) = x(i)+vx(i)*delta_t
      y(i) = y(i)+vy(i)*delta_t
      z(i) = z(i)+vz(i)*delta_t
    ENDDO
    ! Wall recoil module
    IF (it.lt.n_steps) THEN
      ! Compartmentalized confinement in the z direction
      DO i = 1, n_d
        IF (((z(i).le.(-5.*del_z)).and.(vz(i).lt.0.)).or.((z(i).ge.(-
r_d)) &
        .and.(vz(i).gt.0.))) THEN
          IF (tag(i).eq.0) THEN
            vz(i) = -vz(i)
          ENDIF
        ENDIF
      ENDDO
      DO i = n_d+1, n_d+k_b*n_b
        IF (((z(i).le.(r_b+r_d)).and.(vz(i).lt.0.)).or.((z(i).ge.(d-
r_b)) &
        .and.(vz(i).gt.0.))) THEN
          IF (tag(i).eq.0) THEN
            vz(i) = -vz(i)
          ENDIF
        ENDIF
      ENDDO
    ELSE
      ! Full-container confinement in the z direction
      DO i = 1, n_d
        IF (((z(i).le.r_d).and.(vz(i).lt.0.)).or.((z(i).ge.(float(k_b)
*d-r_d)) &
        .and.(vz(i).gt.0.))) THEN
          IF (tag(i).eq.0) THEN
```

```
          vz(i) = -vz(i)
        ENDIF
      ENDIF
    ENDDO
    DO i = n_d+1, n_d+k_b*n_b
      IF (((z(i).le.r_b).and.(vz(i).lt.0.)).or.((z(i).ge.(float(k_b)
*d-r_b)) &
      .and.(vz(i).gt.0.))) THEN
        IF (tag(i).eq.0) THEN
          vz(i) = -vz(i)
        ENDIF
      ENDIF
    ENDDO
  ENDIF
  DO i = 1, n_d+k_b*n_b ! Confinement in the x and y directions
    IF (i.le.n_d) THEN
      r_check = r_d
    ELSE
      r_check = r_b
    ENDIF
    IF (((x(i).le.r_check).and.(vx(i).lt.0.)).or.((x(i).ge.(s-
r_check)) &
    .and.(vx(i).gt.0.))) THEN
      IF (tag(i).eq.0) THEN
        vx(i) = -vx(i)
      ENDIF
    ENDIF
    IF (((y(i).le.r_check).and.(vy(i).lt.0.)).or.((y(i).ge.(s-
r_check)) &
    .and.(vy(i).gt.0.))) THEN
      IF (tag(i).eq.0) THEN
        vy(i) = -vy(i)
      ENDIF
    ENDIF
  ENDDO
  ! Molecular recoil module
  DO j = 1, 1E5
    IF (j.ge.(n_d+k_b*n_b)) THEN
      EXIT
    ENDIF
    DO k = j+1, 1E5
      IF (k.gt.(n_d+k_b*n_b)) THEN
        EXIT
      ENDIF
      dx = x(j)-x(k)
      dy = y(j)-y(k)
      dz = z(j)-z(k)
      dist = dx*dx+dy*dy+dz*dz
      IF (sqrt(dist).gt.(1.5*(r_b+r_d))) THEN
        GOTO 100
      ENDIF
      IF ((j.le.n_d).and.(k.le.n_d)) THEN
        r_check = 2.*r_d
        m_j = m_d
        m_k = m_d
        IF ((sqrt(dist).le.r_check).and.(tag(j).eq.0).and.(tag
(k).eq.0) &
        .and.(time.eq.0.)) THEN
          coll_2d = coll_2d+1 ! Collision counter
        ENDIF
      ENDIF
```

```
        IF ((j.le.n_d).and.(k.gt.n_d)) THEN
          r_check = r_b+r_d
          m_j = m_d
          m_k = m_b
          IF ((sqrt(dist).le.r_check).and.(tag(j).eq.0).and.(tag
(k).eq.0) &
          .and.(time.gt.0.)) THEN
            coll_bd = coll_bd+1
          ENDIF
        ENDIF
        IF ((j.gt.n_d).and.(k.le.n_d)) THEN
          r_check = r_b+r_d
          m_j = m_b
          m_k = m_d
          IF ((sqrt(dist).le.r_check).and.(tag(j).eq.0).and.(tag
(k).eq.0) &
          .and.(time.gt.0.)) THEN
            coll_bd = coll_bd+1
          ENDIF
        ENDIF
        IF ((j.gt.n_d).and.(k.gt.n_d)) THEN
          r_check = 2.*r_b
          m_j = m_b
          m_k = m_b
          IF ((sqrt(dist).le.r_check).and.(tag(j).eq.0).and.(tag
(k).eq.0) &
          .and.(time.gt.0.)) THEN
            coll_2b = coll_2b+1
          ENDIF
        ENDIF
        IF ((sqrt(dist).le.r_check).and.(tag(j).eq.0).and.(tag(k).eq.0))
THEN
          ! Molecular recoil
          IF ((j.le.n_d).and.(k.le.n_d).and.(it.ge.n_steps)) THEN
            EXIT
          ENDIF
          phi = 2.*(dx*(vx(j)-vx(k))+dy*(vy(j)-vy(k))+dz*(vz(j)-vz(k)))/
&
          (dist*(m_j+m_k))
          IF (time.gt.0.) THEN ! Reflection criteria
            IF ((j.gt.n_d).and.(k.le.n_d).and.((vz(k)/(vz(k)
+phi*m_j*dz)).lt. &
            0.)) THEN
              n_ref = n_ref+1
            ENDIF
            IF ((j.le.n_d).and.(k.gt.n_d).and.((vz(j)/(vz(j)-
phi*m_k*dz)).lt. &
            0.)) THEN
              n_ref = n_ref+1
            ENDIF
          ENDIF
          vx(k) = vx(k)+phi*m_j*dx
          vy(k) = vy(k)+phi*m_j*dy
          vz(k) = vz(k)+phi*m_j*dz
          vx(j) = vx(j)-phi*m_k*dx
          vy(j) = vy(j)-phi*m_k*dy
          vz(j) = vz(j)-phi*m_k*dz
          hit(k) = hit(k)+1
          hit(j) = hit(j)+1
          sum_bi = sum_bi+1
        ENDIF
```

```
          IF (sqrt(dist).le.r_check) THEN
            x_j(j) = x(j)+vx(j)*delta_t
            y_j(j) = y(j)+vy(j)*delta_t
            z_j(j) = z(j)+vz(j)*delta_t
            x_k(k) = x(k)+vx(k)*delta_t
            y_k(k) = y(k)+vy(k)*delta_t
            z_k(k) = z(k)+vz(k)*delta_t
            dist = sqrt((x_j(j)-x_k(k))**2.+(y_j(j)-y_k(k))**2.+(z_j(j)-
z_k(k)) &
            **2.)
            IF ((dist.le.r_check).and.(tag(j).eq.0).and.(tag(k).eq.0))
THEN
              tag(j) = 1
              tag(k) = 1
            ELSE IF ((dist.gt.r_check).and.(tag(j).eq.1).and.(tag
(k).eq.1)) THEN
              tag(j) = 0
              tag(k) = 0
            ENDIF
            IF (it.gt.n_steps) THEN
              DO i = 1, n_d
                x_run = x_run+abs(vz(i))
                r_run = r_run+sqrt(vx(i)*vx(i)+vy(i)*vy(i)+vz(i)*vz(i))
                fr = x_run/r_run
              ENDDO
            ENDIF
          ENDIF
          100 CONTINUE
        ENDDO
      ENDDO
      DO i = 1, n_d+k_b*n_b
        IF (hit(i).gt.1) THEN
         sum_ter = sum_ter+1
        ENDIF
        hit(i) = 0
      ENDDO
      IF (n_steps.eq.it) THEN
        j = 0
        DO i = n_d+1, n_d+n_b
          j = j+1
          vxc(j) = vx(i)
          vyc(j) = vy(i)
          vzc(j) = vz(i)
          xc(j) = x(i)
          yc(j) = y(i)
          zc(j) = z(i)
        ENDDO
      ENDIF
      DO i = 1, n_d
        IF (z(i).gt.(float(k_b)*d-r_d)) THEN
          k = 0
          DO j = n_d+k_b*n_b+1, n_d+(k_b+1)*n_b
            k = k+1
            vx(j) = vxc(k)
            vy(j) = vyc(k)
            vz(j) = vzc(k)
            x(j) = xc(k)
            y(j) = yc(k)
            z(j) = zc(k)+float(k_b)*d
          ENDDO
          k_b = k_b+1
```

```
      ENDIF
    ENDDO
    IF (it.ge.n_steps) THEN
      time = time+delta_t
    ENDIF
    it = it+1
    itm = itm+1
    IF (itm.eq.100) THEN
      PRINT *, "  "
      PRINT *, "The time step number is", it
      IF (time.eq.0.) THEN
        PRINT *, "Spin-up phase"
      ELSE
        PRINT *, "Diffusing phase"
      ENDIF
      PRINT *, "<distance> (nm) for the bath molecules is", &
      float(it)*delta_t*v_b
      PRINT *, "<distance> (nm) for the diffusing molecules is", &
      float(it)*delta_t*v_d
      PRINT *, "The cumulative number of bimolecular collisions is ",
sum_bi
      PRINT *, "The cumulative number of termolecular collisions is",
sum_ter
      IF (time.eq.0.) THEN
        PRINT *, "The cumulative number of d-d collisions is", coll_2d
      ENDIF
      PRINT *, "The current number of bins is", k_b
      itm = 0
    ENDIF
  ENDDO
  IF (time.gt.0.) THEN
    PRINT *, "    "
    PRINT *, "The time is", time
    PRINT *, "The average collision frequency for the bath molecules
is", &
    float(coll_2b)/time
    PRINT *, "The average collision frequency between the bath and
diffusing &
    molecules is", float(coll_bd)/time
    IF (coll_bd.gt.0.) THEN
      PRINT *, "The reflection coefficient is", float(n_ref)/float
(coll_bd)
    ENDIF
    PRINT *, "The ratio between the distance traveled in one coordinate
to the &
    total"
    PRINT *, "distance traveled is", fr
    PRINT *, "    "
    PRINT *, "        Bin #        [diffusing molecules]/nm^-3
[Total]"
    DO i = 1, k_b
      n_cd(i) = 0
      n_cb(i) = 0
    ENDDO
    DO j = 1, n_d
      DO i = 1, k_b
        IF ((z(j).ge.(float(i-1)*d)).and.(z(j).le.(float(i)*d))) THEN
          n_cd(i) = n_cd(i)+1
          EXIT
        ENDIF
      ENDDO
```

```
    ENDDO
    DO j = n_d+1, n_d+k_b*n_b
      DO i = 1, k_b
        IF ((z(j).ge.(float(i-1)*d)).and.(z(j).le.(float(i)*d))) THEN
          n_cb(i) = n_cb(i)+1
          EXIT
        ENDIF
      ENDDO
    ENDDO
    DO i = 1, k_b
      PRINT *, i, "             ", float(n_cd(i))/(s*s*d), "             ",
&
      float(n_cd(i)+n_cb(i))/(s*s*d)
    ENDDO
  ENDIF
  n_pair = 0
  DO i = 1, n_d+k_b*n_b-1
    DO j = i+1, n_d+k_b*n_b
      IF ((i.gt.n_d).and.(j.gt.n_d)) THEN
        r_check = 2.*r_b
      ELSE IF ((i.le.n_d).and.(j.le.n_d)) THEN
        r_check = 0.
      ELSE
        r_check = r_b+r_d
      ENDIF
      IF ((sqrt((x(i)-x(j))**2.+(y(i)-y(j))**2.+(z(i)-z(j))**
2.)).le.r_check) &
      THEN
        n_pair = n_pair+1
      ENDIF
    ENDDO
  ENDDO
  PRINT *, "The number of molecules currently in contact are", 2.*float
(n_pair)
  PRINT *, "      "
  PRINT *, "Do you want to continue the simulation (y or n)"
  READ *, answer
  IF (answer.eq."n") THEN
    STOP
  ELSE
    PRINT *, "Input the new number of time steps"
    READ *, steps
  ENDIF
ENDDO

END PROGRAM oned_diff
```