

# MSE627–MD code

Main.f

Common.h  
Parameters.h

Initial Setup,  
Reading the data, ...  
OpenFiles.f, ReadFiles.f, SetInit.f,  
WriteInit.f, Etab-SW.f, Eftab-pair.f,  
Vel.f, SetQuench.f

## Molecular Dynamics

Nord5.f, F-pair.f, F-SW.f,  
Forces.f NbList.f, Quench.f,  
Temper.f, Heating.f, Gather.f,  
Swrite.f, Impact.f

## Clean up after the MD loop

Writing data to disc, ...  
SetEnd.f, SetQuench.f,  
WriteEnd.f

## **MSE 627 References on Molecular Dynamics simulation technique**

1. M. P. Allen, D. J. Tildesley, Computer Simulation of Liquids (Clarendon Press: Oxford, 1990) Call number: QC 145.2 .A43 1990 or QC145.2 .A43 1987
2. D. Frenkel, B. Smit, Understanding Molecular Simulation from Algorithms to Applications (Academic Press: San Diego, 1996) Call number: QD461 .F86 1996
3. Dierk Raabe, Computational materials science: the simulation of materials, microstructures and properties (Wiley-VCH: Weinheim, 1998) Call number: TA403.6 .R23 1998
4. W. G. Hoover, Computational statistical mechanics (Elsevier: Amsterdam, New York, 1991) Call number: QC 174.8.H66 1991
5. K. Binder, Monte Carlo and molecular dynamics simulations in polymer sciences (Oxford University Press: Oxford, New York, 1995) Call number: QD 381.9 .E4 M66 1995
6. M. Metcalf and J. Reid, Fortran 90/95 explained (Oxford University Press: Oxford, New York, 1999) Call number: QA76.73 .F28 M49 1999

## Listing of MSE627-MD code, Main.f, page 1 of 5

```
! Molecular Dynamic Code for use in MSE6270 course
! Supports multi-component systems,
! periodic, free, and rigid boundary conditions,
! Lennard-Jones potential, other potentials can be added as needed
PROGRAM MD
INCLUDE 'common.h'
INTEGER S
!*****
! UNITS:
! LENGTH - 1A
! TIME - 1psec
! MASS - 1amu = 1.66057D-27 Kg
! TEMPERATURE - K
! ENERGY - 1.0364381D-04 eV = 1.66057D-23 J
!*****
! All input/output files that you use should be listed in file md.rc
! UNIT 10 | NAMES OF INPUT/OUTPUT FILES (md.rc) |READ
! UNIT 12 | EXTERNAL IMPACT PARAMETERS |READ
! UNIT 13 | SEEDS FOR THE RUNDOM NUMB. GENERATOR |READ & WRITE
! UNIT 14 | INPUT DATA (MATERIAL AND RUN PARAMETERS) |READ
! UNIT 15 | INPUT COORDINATES AND VELOCITIES |READ
! UNIT 16 | OUTPUT COORDINATES AND VELOCITIES | WRITE
! UNIT 17 | OUTPUT FILE (ENERGY PER ATOM,TEMP., etc.) | WRITE
! UNIT 18 | FILE FOR MAKING SNAPSHOTS (COORDINATES) | WRITE
! UNIT 111| FORCE & ENERGY TABLES (ONLY FOR TESTING) | WRITE
!*****
```

## Listing of MSE627-MD code, Main.f, page 2 of 5

```

!   NAN is total number of particles
!   Ntype is the number of particle types
!   Npots=Ntype*(Ntype+1)/2 is number of types of particle pairs
!   For example, for Ntype=3, Npots=6 and for molecules I of type Ktype(I)
!   and J of type Ktype(J),   IJindex is defined as
!
!       KTYPE(J) = 1:Ntype
!
!       KTYPE(I) = 1:Ntype
!           1 2 3
!           1 3 6
!           2 1 | 3 2 5
!           3 2 | 6 5 4
!           3 2 1
!
!       IJindex(Ktype(I),Ktype(J)) = 1:Npots
!
!*****
! History variable defines properties of the particle other than its type
!
!   KHIST(J)=1 - full dynamics
!   KHIST(J)=2 - temperature control
!   KHIST(J)=3 - rigid
!   KHIST(J)=4 - analytic constraints - dynamic boundary
!*****
character*8 openf
real*8 wcl_st, wcl_end, cpu_st, cpu_end

!   mclock() and rtc() are XL Fortran functions that return CPU and Wall clock time.
!   If your compiler does not support this functions, you can just comment them out.
!   cpu_st=mclock()
!   wcl_st=rtc()
!   Open and read the namelist datafile
!   openf = 'md.rc'
!   CALL OpenFiles(openf)

!   CALL ReadFiles()           ! reading input files

```

Listing of MSE627-MD code, Main.f, page 3 of 5

```
NTYPE=1          ! Number of particle types
Npots=Ntype*(Ntype+1)/2  ! Number of types of particle pairs

! Create energies & forces tables
! Pair potentials and forces are tabulated
IF(KEYBS.EQ.0) Then
  CALL EF1LJ(1,1,1) ! Ar-Ar
ELSEIF (KEYBS.EQ.1) Then
  CALL EF1_SW(1,1,1) ! Si
  CALL EF1_SW(2,2,2) ! Ge
  CALL EF1_SW(3,1,2) ! Si-Ge
ELSE
  Print *, "KEYBS=",KEYBS," is not defined. Program will stop."
  Stop
ENDIF

CALL SetInit(DENSM,DENSN)  ! defining some initial parameters

CALL WriteInit(DENSM,DENSN) ! writing initial output before MD

S=0          ! S - current integration step
HDONE=0      ! HDONE - for HEATING (KFLAG=3)
IF(KFLAG.EQ.1) CALL SetQuench(s)

! Initiate the movie file
! IF(MOV.EQ.2) THEN
!   STMOV=0.0d0
!   CALL Movie()
!   STMV=TIME+STMOV
! ENDIF

CALL NbList()      ! Create a neighbour list

! Fast heating (Through velocity distribution)
IF(KFLAG.EQ.2) CALL Vel()

WRITE(17,*) ' Step Energy Kinetic Potential Temperature'
```

## Listing of MSE627-MD code, Main.f, page 4 of 5

```
main_loop: DO S = 1,NSTEP
  TIME=TIME+DELTA
  CALL Nord5()                ! Integration
!   Updating Neighbour List
  IF (MOD(S,NEWTAB).EQ.0) CALL NbList()
  IF (KFLAG.EQ.1) CALL Quench()    ! Quenching
!   Slow heating of the material
  IF (KFLAG.EQ.3.AND.HDONE.EQ.0) THEN
    CALL Temper(ENT,QINT,POTT,TEMPTR)
    CALL Heating(TEMPTR,HDONE)
  ENDIF
!   Printing output information
  IF (MOD(S,NEPRT).EQ.0.OR.(S.EQ.1)) THEN
    CALL Temper(ENT,QINT,POTT,TEMPTR)
    WRITE(17,9) S,ENT,QINT,POTT,TEMPTR
  ENDIF
!   Collecting data for vibrational spectra calculation
!   CALL Corlfn()
!   Gathering all particles to the initial cell
  IF (MOD(S,NPER).EQ.0) CALL Gather()
!   Writing data for future analysis
  IF (MOD(S,NWRITE).EQ.0.OR.(S.EQ.1)) CALL Swrite()
!   IF (MOV.EQ.3.AND.TIME.GE.STMV) THEN
!     CALL Movie()                ! Write movie file
!     STMV=STMV+STMOV
!   ENDIF
!   External Impact
!   IF (LGOT.EQ.1.AND.Time.LE.ExTime) CALL Impact()
END DO main_loop
```

## Listing of MSE627-MD code, Main.f, page 5 of 5

```
! End of MD loop

CALL SetEnd()

IF (KFLAG.EQ.1) CALL SetQuench(S)

cpu_end=mclock()
wcl_end=rtc()
Write(17,*) 'Wall clock time of main: ',wcl_end-wcl_st,' sec'
Write(17,*) 'CPU time used in main: ',(cpu_end-cpu_st)/100.0d0,' sec'

CALL WriteEnd()

9 FORMAT(I5,3(1X,D11.5),1X,F6.0)

STOP
END
```

## Listing of MSE627-MD code, common.h, page 1 of 2

```
IMPLICIT REAL*8(A-H,O-Z)
character*14 drname
logical FullList
INCLUDE 'parameters.h'

! Coordinates, velocities, and forces
COMMON/XXX/ X(LPMX3),Q1(LPMX3),F(LPMX3)

! Coordinates of the center of the computational cell
COMMON/CENTR/ XCENTR,YCENTR,ZCENTR

! Energies per atom (total, potential, and kinetic)
! VW - kinetic energy at the previous step (used in quenching)
COMMON/ENY/ EN(LPMX),POT(LPMX),QIN(LPMX),VW(LPMX)

! Force and Energy tables for pair potentials
COMMON/TTT/ FT(KPMX,NT),UT(KPMX,NT)

! Index that defines type of the pair potential for two atoms
COMMON/IND/ IJINDEX(KTMX,KTMX)

! Cutoff distances for pair potentials and neighbor lists
COMMON/CUT/ DXR(KPMX),RM(KPMX),RList(KPMX),RList2(KPMX),Rskin

! Parameters for 3-body term of SW
COMMON/SW/ SW_sig(KPMX),SW_lam(KPMX),SW_gam(KPMX),SW_eps(KPMX)

! Neighbor lists arrays
COMMON/NBR/ NNG(LPMX),NNNG(MAXNNB,LPMX),FullList

! Characteristics of the atoms
COMMON/KTP/ KTYPE(LPMX),KRIGID(LPMX),KHIST(LPMX),KTT(KTMX)

! Masses of the atoms
COMMON/MSS/ XMASS(KTMX),G1(KTMX)
```

## Listing of MSE627-MD code, common.h, page 2 of 2

! Higher time derivatives of the coordinates and parameters for  
! Nordsieck integrator  
COMMON/NORD/ Q2(LPMX3),Q3(LPMX3),Q4(LPMX3),Q5(LPMX3), &  
C1,C2,C3,C4,C5

! Sizes of the computational cell  
COMMON/CELL/ XL,XLHALF,YL,YLHALF,ZL,ZLHALF

! Input parameters  
COMMON/NNN/ NAN,NAN3,NO1,NRIGID,NSTEP,LIDX,LIDY,LIDZ,KBOUND, &  
NDIM,NEWTAB,NEPRT,NWRITE,NPER,NTYPE,NPOTS,MOV,KFLAG,LGOT  
COMMON/ExF/ ExForce, ExTime  
COMMON/DDD/ TIME,QTEM,DELTA

! Variables used to define names of input/output files  
COMMON/dr1/ LDR  
COMMON/dr2/ DRNAME

! Seed for random number generator  
COMMON/RNDN/ ISEED(64)

! This variables can be used to create animations  
COMMON/MOV/ stmov, radius(100),rcolor(100),gcolor(100),bcolor(100)

! It is convenient to have coordinates, velocities, and forces  
! in both two- and one dimensional arrays  
DIMENSION XD(3,LPMX), FD(3,LPMX), &  
Q1D(3,LPMX),Q2D(3,LPMX),Q3D(3,LPMX),Q4D(3,LPMX),Q5D(3,LPMX)  
EQUIVALENCE (X(1),XD(1,1)), (F(1),FD(1,1)), &  
(Q1(1),Q1D(1,1)),(Q2(1),Q2D(1,1)),(Q3(1),Q3D(1,1)), &  
(Q4(1),Q4D(1,1)),(Q5(1),Q5D(1,1))

## Listing of MSE627-MD code, parameters.h

Parameter (LPMX = 5000) ! Maximum number of particles  
Parameter (LPMX3 = LPMX\*3) !  
Parameter (KTMX = 3) ! Maximum number of particle types  
Parameter (KPMX = KTMX\*(KTMX+1)/2) ! Maximum number of potential types  
Parameter (MAXNNB = 150) ! Max. number of neighbour pairs  
Parameter (NT = 2000) ! Points in the energy & force tables

!---- Fundamental constants -----

Parameter (PI = 3.1415926535897932D+00) !  
Parameter (EVTOJOU = 1.60219D-19) ! J/eV  
Parameter (AMUTOKG = 1.6605402D-27) ! kg/amu  
Parameter (BK = 8.617385D-05) ! Boltzman constant, eV/K  
Parameter (XJOUTOEV = 1.d0/EVTOJOU) ! eV/J  
Parameter (CSPEED = 2.99792458D+08) ! speed of light in m/s  
Parameter (HPLANCK = 6.6260755D-34) ! Planck's constant in J\*s

!-----

!---- Transfer to program units -----

Parameter (ENUNIT = AMUTOKG\*1.d4\*XJOUTOEV) ! eV/pr.u.

!-----

## Listing of MSE627-MD code, input files

### md.rc:

UNIT 12	md.impact	old
UNIT 13	md.random	old
UNIT 14	md.input	old
UNIT 15	Ar.data	old
UNIT 16	Art.data	unknown
UNIT 17	Art.out	unknown
UNIT 18	Art.xyz	unknown
*DIR 01	data	dirname

### md.input:

5000	- NSTEP (Number of steps)
50	- NEWTABL (Step of neighbors list renewal)
100	- NEPRT (Step of printing output information)
500	- NWRITE (Step of writing output information)
200	- NPER (Step of gathering molecules to the comp. cell)
0	- MOV (1 - Write snapshot; 2&3 – movie file)
2	- KFLAG (1-Quench, 2-Velocity distribution (Vel.f), 3-Heating.f)
0	- KEYBS (0-pair potential,1-Stillinger Weber)
1	- LIDX (1- X periodic,0-free boundary conditions])
1	- LIDY (1- Y periodic, 0-free boundary conditions])
1	- LIDZ (1-On [Z PBC],0-Off [e.g. XY PBC,Z - boundary layer])
0	- KBOUND (Type of boundary 0-free, 1-rigid, 2-...)
0	- LGOT (0-Impact is switched off, 1-on)
3	- NDIM (3 - 3D simulation, 2 - 2D simulation)
200.00000	- QTEM (Temperature distributed by VEL)
0.01000	- DELTA (Timestep of integration in pr.unit [psec])
2.00000	- RSkin (Skin depth for neighbor list [A])

## Listing of MSE627-MD code, input files

### **5Ar.data:**

```
5 0.0
20.0 20.0 20.0 0.0 0.0 0.0
1 1 1 1 1
1 0.0 0.0 -7.644
1 0.0 0.0 -3.822
1 0.0 0.0 0.0
1 0.0 0.0 3.822
1 0.0 0.0 7.644
0.0 0.0 0.0
0.0 0.0 0.0
0.0 0.0 0.0
0.0 0.0 0.0
0.0 0.0 0.0
```

### **md.random:**

```
1104724792 2017460224
220.795516974250006 27.9135227209889543
```

### **md.impact:**

```
100.00000 - ExForce (Force per unit area [N/A2])
100.00000 - ExTime (Duration of the external impact [psec])
```

## Listing of MSE627-MD code, OpenFiles.f

! THIS SUBROUTINE OPENS MOST OF THE INPUT/OUTPUT FILES

```
SUBROUTINE OpenFiles(openf)
character*14 str1, str2
character*4 str
character*8 openf
INCLUDE 'common.h'
OPEN (UNIT = 10,FILE=openf)

opener_loop: DO
    read(10,1001,end=1004) str,iunit,str1,str2
1001  FORMAT(A4,1x,I2,1x,A14,1x,A7)
    IF (str(1:1).eq. '#') CYCLE opener_loop
    IF (str(1:1).eq. '*') THEN
        ldr=leng(str1)
        IF(ldr.eq.0) Then
            WRITE(*,*) 'Wrong name of directory',str1
            STOP
        ENDIF
        drname(1:ldr)=str1(1:ldr)
        CYCLE opener_loop
    ENDIF
    lf1=leng(str1)
    IF (lf1.eq.0) Then
        WRITE(*,*) 'Wrong format in the file ',openf
        STOP
    ENDIF
    lf2=leng(str2)
    OPEN (iunit,file=str1(1:lf1),status=str2(1:lf2),err=255)
    CYCLE opener_loop
255  WRITE(*,*) 'Error while reading file ',str1
    STOP
1004  EXIT opener_loop                                RETURN
    END DO opener_loop                                END
```

## Listing of MSE627-MD code, ReadFiles.f, page 1 of 2

```
! THIS SUBROUTINE READS MOST OF THE INPUT FILES

SUBROUTINE ReadFiles()
INCLUDE 'common.h'

! Read Input Coordinates/Velocities File
REWIND 15
READ(15,*) NAN,TIME
If (NAN.gt.LPMX) then
    write(*,*) ' NAN=',NAN,' is greater than LPMX=',LPMX, ' Program stops.'
    stop
Endif
READ(15,*) XL,YL,ZL,XCENTR,YCENTR,ZCENTR
READ(15,*) (KTYPE(J),J=1,NAN)
READ(15,*) (KHIST(J),XD(1,J),XD(2,J),XD(3,J),J=1,NAN)
READ(15,*) (Q1D(1,J),Q1D(2,J),Q1D(3,J),J=1,NAN)

! READ INPUT DATA
REWIND 14
READ(14,100) NSTEP,NEWTAB,NEPRT,NWRITE,NPER,MOV, &
    KFLAG, KEYBS,LIDX,LIDY,LIDZ,KBOUND,LGOT,NDIM
READ(14,200) QTEM, DELTA, RSkin
```

## Listing of MSE627-MD code, ReadFiles.f, page 2 of 2

```
! READ INPUT DATA FOR EXTERNAL IMPACT
! REWIND 12
! READ(12,200) ExForce, ExTime

! READ RANDOM NUMBERS FOR VEL SUBROUTINE
CALL Random_seed(SIZE = Kseed)
REWIND 13
READ(13,*) (ISEED(i), i=1,Kseed) ! for random_number(rww1)
READ(13,*) U1,U2 ! for rn()
CALL Random_seed(put = iseed)
! CALL Random_seed(generator = 1)

100 FORMAT (I10)
200 FORMAT (D15.5)
RETURN
END
```

Listing of MSE627-MD code, SetInit.f, page 1 of 3

```
! THIS SUBROUTINE SETS THE INITIAL PARAMETERS
SUBROUTINE SetInit(DENSM,DENSN)
INCLUDE 'common.h'

! Counting particles of different types
KTT(1:NTYPE)=0
DO I=1,NAN
    DO J=1,NTYPE
        IF(KTYPE(I).EQ.J) KTT(J)=KTT(J)+1
    ENDDO
ENDDO

KTOT=0
DO J=1,NTYPE
    KTOT=KTOT+KTT(J)
ENDDO

IF (KTOT.NE.NAN) Then
    Print *,'KTYPE(I), NTYPE, and NAN do not match, Program stops'
    STOP
ENDIF

! Density calculation
VOLUME=(XL*1.0D-08)*(YL*1.0D-08)*(ZL*1.0D-08)      ! Volume in cm3
TMASS=0.0d0
DO J=1,NTYPE
    TMASS=TMASS+KTT(J)*XMASS(J)                    ! Total mass in amu
ENDDO
DENSM=TMASS*AMUTOKG*1.0D+03/VOLUME                 ! gr/cm3
DENSN=NAN/VOLUME                                   ! molecules/cm3
```

## Listing of MSE627-MD code, SetInit.f, page 2 of 3

```
! Let's define a few variables
NO1=NAN-1
NAN3=NAN+NAN+NAN
XLHALF=XL*0.5d0
YLHALF=YL*0.5d0
ZLHALF=ZL*0.5d0

! We will use DXR(I) to pick a value of force or energy that
! corresponds to a given interparticle distance from the table
DO I=1, NPOTS
    DXR(I)=NT/RM(I)
    RLIST(I) = RM(I)+RSKIN
    RLIST2(I) = RLIST(I)*RLIST(I)
ENDDO

FullList = .FALSE.           ! for pair potential
IF(KEYBS.EQ.1) FullList = .TRUE. ! for MC or S-W potential

! Let's assume that we have up to 3 types of atoms interacting via up to 6 pair potentials
IJINDEX(1,1)=1
IJINDEX(1,2)=3
IJINDEX(1,3)=6
IJINDEX(2,1)=3
IJINDEX(2,2)=2
IJINDEX(2,3)=5
IJINDEX(3,1)=6
IJINDEX(3,2)=5
IJINDEX(3,3)=4
```

## Listing of MSE627-MD code, SetInit.f, page 3 of 3

```
! Defining the boundary layer
NRIGID=0
DO I=1,NAN
    VW(I)=0.0d0
    KRIGID(I)=0
    IF(KHIST(I).EQ.3) KRIGID(I)=KBOUND
    IF(KHIST(I).EQ.3) NRIGID=NRIGID+1
ENDDO

! It is convenient to multiply velocities by timestep DELTA
! when a predictor-corrector algorithm is used.
DO I=1,NAN3
    Q1(I)=Q1(I)*DELTA
    Q2(I)=0.0d0
    Q3(I)=0.0d0
    Q4(I)=0.0d0
    Q5(I)=0.0d0
ENDDO

! G1 is used in Nordsieck Integration method
DO J=1,NTYPE
    G1(J)=0.5d0*DELTA*DELTA/XMASS(J)
ENDDO

! Nordsieck predictor-corrector coefficients
C1=3.d0/20.d0
C2=251.d0/360.d0
C3=11.d0/18.d0
C4=1.d0/6.d0
C5=1.d0/60.d0

RETURN
END
```

Listing of MSE627-MD code, WriteInit.f, page 1 of 2

```
! THIS SUBROUTINE WRITES INITIAL INFORMATION FOR THE RUN
SUBROUTINE WriteInit(DENSM,DENSN)
INCLUDE 'common.h'

WRITE(17,15) NSTEP,DELTA,KFLAGKEYBS,,LIDX,LIDY,LIDZ,KBOUND, &
    NDIM,QTEM,NEWTAB,NEPRT,NWRITE,NPER,MOV,NT,RSkin
15 FORMAT( 6X,' -----> MD RUN:  '/' &
    'NSTEP=',I5,' (Number of steps);/' &
    'DELTA=',F7.5,' (Integration time step in [psec]);',/, &
    'KFLAG=',I1,' (1-Quench, 2-Vel ....);',/, &
    'KEYBS=',I1,' (0-pair potential, 4-Stillinger Weber);',/, &
    'LIDX=',I1,' (1-periodicity in X direct.,0-free boundaries);/' &
    'LIDY=',I1,' (1-periodicity in Y direct.,0-free boundaries);/' &
    'LIDZ=',I1,' (1-Z PBC,0-Z free or boundary layer);/' &
    'KBOUND=',I2,' (0-Free,1-rigid, 2-...);/' &
    'NDIM=',I2,' (3 - 3D simulation,2- 2D simulation);/' &
    'QTEM=',F6.1,' (Temperature distributed by VEL,[K]);/' &
    'NEWTAB=',I3,' (Step of neighbors list renewal);/' &
    'NEPRT=',I4,' (Step of printing output information);/' &
    'NWRITE=',I5,' (Step of writing output information);/' &
    'NPER=',I4,' (Step of gathering molecules to the comp. cell);/' &
    'MOV=',I4,' (1 - Write snapshots, 2 - movie file);/' &
    'NT=',I4,' (Number of points in the energy & force tables);/' &
    'RSkin=',F6.3,' (Skin depth in neighbor list calculation [A]);/'

WRITE(17,17) XL,YL,ZL,DENSM,DENSN,NAN
17 FORMAT( 6X,' -----> MATERIAL:  '/' &
    'XL=',D11.5,' (X size of the computational cell [A]);/' &
    'YL=',D11.5,' (Y size of the computational cell [A]);/' &
    'ZL=',D11.5,' (Z size of the computational cell [A]);/' &
    'DENSM=',D11.5,' (Density of the material [gm/cm3]);/' &
    'DENSN=',D11.5,' (Density of the material [molec./cm3]);/' &
    'NAN=',I6,' (Number of particles in the computational cell);/' &
    6X,' -----> TYPES OF PARTICLES:  ')
```

```
WRITE(17,18) (I,KTT(I),XMASS(I),I=1,NTYPE)
18 FORMAT('Type ',I2,': ',I5,' particles. XMASS =',D11.5,' [amu].')

WRITE(17,16) LGOT,ExForce,ExTime
16 FORMAT( 6X,' -----> EXTERNAL IMPACT:  '/' &
          'LGOT=',I1,' (0-External impact is off, 1 - on);/' &
          'ExForce=',D11.5,' (External Force [N]);/' &
          'ExTime=',D11.5,' (External Impact Duration [psec]);/')

! CALL FLUSH(17)

RETURN
END
```

```
! THIS SUBROUTINE SETS THE SYSTEM FOR QUENCHING
SUBROUTINE SetQuench(S)
INCLUDE 'common.h'
INTEGER S

! Stop the movement
Q1(1:NAN3)=0.0d0
Q2(1:NAN3)=0.0d0
Q3(1:NAN3)=0.0d0
Q4(1:NAN3)=0.0d0
Q5(1:NAN3)=0.0d0

! It is convenient for further analysis to have particles ordered in some way
CALL Reorder()
TIME=0.0d0

IF(S.EQ.0) THEN
! Geometrical center of the computational cell
XMAX=-1.d6
YMAX=-1.d6
ZMAX=-1.d6
XMIN=1.d6
YMIN=1.d6
ZMIN=1.d6
DO 1 I=1,NAN
    XMAX=DMAX1(XMAX,XD(1,I))
    YMAX=DMAX1(YMAX,XD(2,I))
    ZMAX=DMAX1(ZMAX,XD(3,I))
    XMIN=DMIN1(XMIN,XD(1,I))
    YMIN=DMIN1(YMIN,XD(2,I))
1    ZMIN=DMIN1(ZMIN,XD(3,I))
```

```
XCEN1R1=XMIN+(XMAX-XMIN)/2.0d0
YCEN1R1=YMIN+(YMAX-YMIN)/2.0d0
ZCEN1R1=ZMIN+(ZMAX-ZMIN)/2.0d0
if (XCEN1R1.NE.XCEN1R) write (*,*) 'XCEN1R is moved from ', &
    XCEN1R,' to ',XCEN1R1
if (YCEN1R1.NE.YCEN1R) write (*,*) 'YCEN1R is moved from ', &
    YCEN1R,' to ',YCEN1R1
if (ZCEN1R1.NE.ZCEN1R) write (*,*) 'ZCEN1R is moved from ', &
    ZCEN1R,' to ',ZCEN1R1
XCEN1R=XCEN1R1
YCEN1R=YCEN1R1
ZCEN1R=ZCEN1R1
ENDIF

RETURN
END
```

## Listing of MSE627-MD code, Reorder.f

! Reordering of the particles in the computational cell. Particles are being reordered by their  
! z coordinate starting from zmin. This is not the most efficient way to reorder, but we  
! are doing this only once, and the efficiency is not so important here.

```
SUBROUTINE Reorder()
INCLUDE 'common.h'
M1=NAN-1
M2=M1
outer_loop: DO J=1,M1
  inner_loop: DO I=1,M2
    test = XD(3,i+1)-XD(3,i)
    If (test.LT.0.0d0) then
      Rf=XD(3,I+1)
      XD(3,I+1)=XD(3,I)
      XD(3,I)=Rf

      ... Same for XD(2,I), XD(1,I), KHIST(I), KTYPE(I) ...

    END IF
  END DO inner_loop
  M2=M2-1
END DO outer_loop
RETURN
END
```

Listing of MSE627-MD code, NbList.f, page 1 of 2

```
SUBROUTINE NbList()
INCLUDE 'common.h'

!SMP$ SCHEDULE (affinity)
nc=0
NNG(1:NAN)= 0 ! contains the number of neighbors of particle i

!SMP$ ASSERT(NODEPS)
!SMP$ PARALLEL DO &
!SMP$& PRIVATE(I,I3,IP1,J,J3,K,DX,DY,DZ,RR), &
!SMP$& SHARED(RList,RList2,LIDX,LIDY,LIDZ,XL,YL,ZL, &
!SMP$& XLHALF,YLHALF,ZLHALF,R,X,NNNG,NNG,NAN), &
!SMP$& REDUCTION(NC)
outer_loop: DO I=1,NAN
    ktypei=KTYPE(I)
    IP1=I+1
    inner_loop: DO J=IP1,NAN
        IJ = ijindex(KTYPEI, KTYPE(J))

        DZ=XD(3,I)-XD(3,J)
        IF (LIDZ.EQ.1) THEN
            IF (DZ.GT.ZLHALF) DZ=DZ-ZL
            IF (DZ.LT.-ZLHALF) DZ=DZ+ZL
        ENDIF

        IF (ABS(DZ).LT.RList(IJ)) THEN
            DX=XD(1,I)-XD(1,J)
            IF (LIDX.EQ.1) THEN
                IF (DX.GT.XLHALF) DX=DX-XL
                IF (DX.LT.-XLHALF) DX=DX+XL
            ENDIF
```

Listing of MSE627-MD code, NbList.f, page 2 of 2

```
IF (ABS(DX).LT.RList(IJ)) THEN
    DY=XD(2,I)-XD(2,J)
    IF (LIDY.EQ.1) THEN
        IF (DY.GT.YLHALF) DY=DY-YL
        IF (DY.LT.-YLHALF) DY=DY+YL
    ENDIF

    IF (ABS(DY).LT.RList(IJ)) THEN
        RR=DX*DX+DY*DY+DZ*DZ
        IF (RR.LT.RList2(IJ)) THEN
!           We've found a neighbor close enough to be on the list
            NNG(I)=NNG(I)+1    ! increase the count of neighbors
            NNNG(NNG(I),I)=J    ! add the neighbor of atom I

!           Some potentials (e.g. SW,EAM) need the complete list
            full: IF (FullList) THEN
                NNG(J)=NNG(J)+1
                NNNG(NNG(J),J)=I
            END IF full

        END IF
    END IF
END IF
END IF
END IF
END DO inner_loop
IF (NNG(I).gt.MAXNNB) THEN
    print *, ' TOO MANY NEIGHBOURS: ',NNG(I),' > ', maxnnb
    STOP
ENDIF
nc=nc+nng(i)
END DO outer_loop
RETURN
END
```

## Listing of MSE627-MD code, Eftab-pair.f, page 1 of 2

- ! Making the Energy and Force tables
- ! Number of pair potential defined here should be
- !  $N_{\text{pots}} = N_{\text{type}} * (N_{\text{type}} + 1) / 2$ ,  $N_{\text{type}}$  is number of particle types
- ! Right now only Lennard - Jones is defined. You can
- ! add more subroutines as needed for your term project.

```
SUBROUTINE EF1LJ(KTEF,KE1,KE2)
```

```
INCLUDE 'common.h'
```

- ! KTEF - type of the potential
- ! KE1, KE2 - elements from the list Element(N)
- ! LJeps is in eV, LJsig is in A, LJmass is in amu
- ! [D.V.Matyushov and R. Schmid, J.Chem.Phys. 104, 8627 (1996)] - Ar
- ! [T.Halicioglu and G.M.Pound, Phys.Stat.Sol.A 30, 619 (1975)] - metals
- ! Cross-species parameters are fixed using the Lorentz-Berthelot rules
- ! [see, e.g. J. S. Rowlinson, Liquid and liquid mixtures (Butterworth Scientific, London, 1982)]

```
REAL*8 LJCeps, LJCsig
```

```
CHARACTER*2, DIMENSION(7), PARAMETER:: Element = &  
(/'Ar','Au','Ni','Pd','Pt','Ag','Cu'/)
```

```
REAL*8, DIMENSION(7), PARAMETER:: LJeps= &  
(/0.0103d0, 0.4415d0, 0.5197d0, 0.4267d0, 0.6817d0, 0.3448d0, 0.4094d0/)
```

```
REAL*8, DIMENSION(7), PARAMETER:: LJsig= &  
(/3.405d0, 2.637d0, 2.282d0, 2.52d0, 2.542d0, 2.644d0, 2.338d0/)
```

```
REAL*8, DIMENSION(7), PARAMETER:: LJmass= &  
(/40.0d0, 197.0d0, 58.7d0, 106.4d0, 195.1d0, 107.9d0, 63.6d0/)
```

```
Write (17,*) 'Creating tables for ', Element(KE1),'-',Element(KE2),' potential'
```

Listing of MSE627-MD code, Eftab-pair.f, page 2 of 2

```
IF(KE1.EQ.KE2) Then
  LJCeps=LJeps(KE1)
  LJCsig=LJsig(KE1)
  XMass(KTEF) = LJmass(KE1)  ! Mass of the particle in pr.unit [aem]
  RM(KTEF) = 2.5*LJCsig      ! Cutoff distance for interaction potential [A]
ELSE
  LJCeps=SQRT(LJeps(KE1)*LJeps(KE2))
  LJCsig=(LJsig(KE1)+LJsig(KE2))/2.0d0
  RM(KTEF) = 2.5*LJCsig
ENDIF

!W Uncomment this line if want to plot the potential
!W OPEN (UNIT = 111,FILE='LJ-'/Element(KE1)'/-'/Element(KE2)'/'.pot')
sig6 = LJCsig**6
DX=RM(KTEF)/NT
XT=0.0d0
DO I=1,NT
  XT=XT+DX
  XT6=XT**6
  sr6 = sig6/XT6                ! (sigma/r)**6
  UT(KTEF,I) = 4.0d0 * LJCeps * sr6 * (sr6 - 1.0d0)  ! potential
  FT(KTEF,I) = LJCeps * 48.0d0/XT * sr6 * (sr6 - 0.5d0) ! force
!   It is a common practice to tabulate force/r rather than force
!   But I am not doing this here to make code more transparent
!W Uncomment this line if want to plot the potential
!W WRITE (111,*) XT,UT(KTEF,I),FT(KTEF,I)
!   Transfer to program units
  UT(KTEF,I) = UT(KTEF,I)/ENUNIT
  FT(KTEF,I) = FT(KTEF,I)/ENUNIT
END DO

!W Uncomment this line if want to plot the potential
!W CLOSE (UNIT = 111)
RETURN    END
```

## Listing of MSE627-MD code, Vel.f, page 1 of 2

! This subroutine distributes velocities to NAN particles in order to make even  
! distribution of kinetic energies from 0 to 6kT. Half of this energy will be transferred  
! to the potential energy. (I Assume that the initial configuration is quenched [POT.EN.=0])  
! You can also read about initialization of velocities in Frenkel & Smit p 56-57

```
SUBROUTINE Vel()
```

```
INCLUDE 'common.h'
```

```
DIMENSION V(NAN)
```

```
xyz_loop: DO IXYZ=1,3
```

```
    SK=0.0d0
```

```
    QIN1=0.0d0
```

```
    DO I=1,NAN
```

```
        call random_number(rww)
```

```
        V(I)=2.*DELTA*SQRT(QTEM*BK*RWW/ENUNIT/XMASS(KTYPE(I)))
```

```
        call random_number(rww1)
```

```
        AWW=RWW1-0.5
```

```
        V(I)=SIGN(V(I),AWW)
```

```
        SK=SK+RWW
```

```
    END DO
```

! FINITE SIZE CORRECTION

```
P=NAN/2.0d0/SK
```

```
VM=0.0d0
```

```
DO I=1,NAN
```

```
    V(I)=V(I)*P
```

```
    VM=VM+V(I)*XMASS(KTYPE(I))
```

```
END DO
```

! TOTAL MOMENTUM CORRECTION

```
VM=VM/NAN
```

```
DO I=1,NAN
```

```
    V(I)=V(I)-VM/XMASS(KTYPE(I))
```

```
    QIN1=QIN1+V(I)*V(I)*XMASS(KTYPE(I))/2.0d0
```

```
END DO
```

```
!      TEMPERATURE CORRECTION
      T1=QIN1*ENUNIT/BK/NAN/DELTA/DELTA
      WRITE(6,*) 'VEL-----> T=',QTEM,', Tvel=',T1,' KELVIN.'
      PR=SQRT(QTEM/T1)
      QIN1=0.0d0
      DO I=1,NAN
          V(I)=V(I)*PR
          QIN1=QIN1+V(I)*V(I)*XMASS(KTYPE(I))/2.0d0
      END DO
      T2=QIN1*ENUNIT/BK/NAN/DELTA/DELTA
      WRITE(6,*) 'VEL-----> T=',QTEM,',Corrected Tvel=',T2,' KELVIN.'

!      Now let's distribute the velocities to Q1D(1:3, 1:NAN)
      DO I=1,NAN
          Q1D(IXYZ,I)=V(I)
      ENDDO
END DO xyz_loop

RETURN
END
```

## Listing of MSE627-MD code, Nord5.f, page 1 of 2

```
! Nordsieck Integrator (fifth-order predictor-corrector algorithm)
! [Allen & Tildesley, p.340], [Lecture notes for MSE 524]
! Kinetic energy is calculated here as well.
SUBROUTINE NORD5()
  INCLUDE 'common.h'
  !SMP$ SCHEDULE (affinity)
  !SMP$ ASSERT(NODEPS)
  !SMP$ PARALLEL DO &
  !SMP$& PRIVATE(I,I3,P1,P2,P3,P4,P5,P6,P7,P8,P9), &
  !SMP$& SHARED(X,Q1,Q2,Q3,Q4,Q5,KRIGID,NAN3)
  trans_loop: DO I=1,NAN3
    IF(NDIM.EQ.2.AND.MOD(I,3).EQ.0) CYCLE trans_loop
    I3=(I-0.5)/3.+1.
    IF (KRIGID(I3).EQ.1) CYCLE trans_loop
    P1=Q2(I)
    P2=Q3(I)
    P3=P2+P2+P2
    P4=Q4(I)
    P5=P4+P4
    P6=P5+P5
    P7=Q5(I)
    P8=(P7+P7+P7)+(P7+P7)
    P9=P8+P8
    X(I)=P7+P4+P2+P1+Q1(I)+X(I)
    Q1(I)=P8+P6+P3+P1+P1+Q1(I)
    Q2(I)=P9+P6+P5+P3+P1
    Q3(I)=P9+P6+P2
    Q4(I)=P8+P4
  END DO trans_loop
```

```
CALL Forces()
```

```
QIN(1:NAN)=0.0d0
```

```
!SMP$ ASSERT(NODEPS)
```

```
!SMP$ PARALLEL DO &
```

```
!SMP$& PRIVATE(I,I3,GG,P), &
```

```
!SMP$& SHARED(KRIGID,G1,REQSTR,KTYPE,F,Q1,Q2,Q3,Q4,Q5, &
```

```
!SMP$& C1,C2,C3,C4,C5,NAN3,NPU,NBS), &
```

```
!SMP$& REDUCTION(QIN)
```

```
trans2_loop: DO I=1,NAN3
```

```
  IF(NDIM.EQ.2.AND.MOD(I,3).EQ.0) CYCLE trans2_loop
```

```
  I3=(I-0.5)/3.+1.
```

```
  IF(KRIGID(I3).EQ.1) CYCLE trans2_loop
```

```
  GG=G1(KTYPE(I3))
```

```
  P=GG*F(I)-Q2(I)
```

```
  X(I)=X(I)+C1*P
```

```
  Q1(I)=Q1(I)+C2*P
```

```
  Q2(I)=Q2(I)+P
```

```
  Q3(I)=Q3(I)+C3*P
```

```
  Q4(I)=Q4(I)+C4*P
```

```
  Q5(I)=Q5(I)+C5*P
```

```
  QIN(I3)=QIN(I3)+Q1(I)*Q1(I)*0.25/GG
```

```
END DO trans2_loop
```

```
RETURN
```

```
END
```

## Listing of MSE627-MD code, Forces.f

```
! Force calculation
SUBROUTINE Forces()
INCLUDE 'common.h'

F(1:NAN3)=0.0d0
POT(1:NAN)=0.0d0

IF(KEYBS.EQ.0) Then
  CALL F_pair()
ELSEIF (KEYBS.EQ.1) Then
  CALL F_SW()
ENDIF

RETURN
END
```

## Listing of MSE627-MD code, F-pair.f, page 1 of 2

```
! Evaluation of forces for pair potentials with neighbor list
! Multiple potentials can be used. Potential energy is calculated.
! Frenkel & Smit, p.59
SUBROUTINE F_pair()
INCLUDE 'common.h'
LOGICAL :: keySTOP=.FALSE.
outer_loop: DO I=1,NAN
  ktypei=KTYPE(I)
  inner_loop: DO K=1,NNG(I)
    J=NNNG(k,i)
    DX=XD(1,J)-XD(1,I)
    DY=XD(2,J)-XD(2,I)
    DZ=XD(3,J)-XD(3,I)
! A more elegant implementation of periodic boundary condition
! DX=DX-XL*NINT(DX/XL) is time consuming because of the nearest
! integer function NINT() in FORTRAN. See the footnote on p. 59
! of Frenkel & Smit, about the inefficiency of NINT
! Periodicity *****
    IF (LIDX.EQ.1) THEN
      IF (DX.GT.XLHALF) DX=DX-XL
      IF (DX.LT.-XLHALF) DX=DX+XL
    ENDIF
    IF (LIDZ.EQ.1) THEN
      IF (DZ.GT.ZLHALF) DZ=DZ-ZL
      IF (DZ.LT.-ZLHALF) DZ=DZ+ZL
    ENDIF
    IF (LIDY.EQ.1) THEN
      IF (DY.GT.YLHALF) DY=DY-YL
      IF (DY.LT.-YLHALF) DY=DY+YL
    ENDIF
! End of periodicity *****
```

Listing of MSE627-MD code, F-pair.f, page 2 of 2

```
IJ = ijindex(KTYPEI, KTYPE(J))
RR=DX*DX+DY*DY+DZ*DZ
RD=SQRT(RR)
IF (RD.GE.RM(IJ)) CYCLE inner_loop

!
If your run crashes, you can try to uncomment this part
IF (RD.LE.RM(IJ)/NT) THEN
    Print *, 'Molecules ',I,' and ',J,' are too close. Program will stop.'
    keySTOP=.TRUE.
ENDIF

P=RD*D XR(IJ)
KF=P
P=P-KF

!
Potential energy (give half the potential to each particle)
PENER=.5*(UT(IJ,KF)+(UT(IJ,KF+1)-UT(IJ,KF))*P)
POT(I)=POT(I)+PENER
POT(J)=POT(J)+PENER

P=FT(IJ,KF)+(FT(IJ,KF+1)-FT(IJ,KF))*P
P=P/RD
DX=P*DX           ! x component of force
FD(1,I)=FD(1,I)-DX
FD(1,J)=FD(1,J)+DX
DZ=P*DZ           ! z component of force
FD(3,I)=FD(3,I)-DZ
FD(3,J)=FD(3,J)+DZ
DY=P*DY           ! y component of force
FD(2,I)=FD(2,I)-DY
FD(2,J)=FD(2,J)+DY

    End do inner_loop
End do outer_loop
if (keySTOP) STOP

RETURN
END
```

## Listing of MSE627-MD code, Quench.f

! This is the fastest way to stop motions in the system and to quench it down to the  
! minimum (local or global) of the potential surface.

```
SUBROUTINE Quench()
```

```
INCLUDE 'common.h'
```

! We stop atom I when its kinetic energy QIN(I) starts to decrease.

! VW(I) is the kinetic energy of the atom I from the previous step.

```
trans_loop: DO I=1,NAN
```

```
  IF (QIN(I).GE.VW(I)) Then
```

```
    VW(I)=QIN(I)
```

```
  ELSE
```

```
    VW(I)=.0
```

```
    Q1D(1,I)=0.0d0
```

```
    Q1D(2,I)=0.0d0
```

```
    Q1D(3,I)=0.0d0
```

```
    QIN(I)=0.
```

```
  END IF
```

```
end do trans_loop
```

```
RETURN
```

```
END
```

## Listing of MSE627-MD code, Temper.f

- ! Calculation of temperature and averaged energies. Energies in eV/particle, Temp. in K.
- ! You can modify this to calculate temperature for atoms with KHIST(J)=2 only.

```
SUBROUTINE TEMPER(ENT,QINT,POTT,TEMPTR)
```

```
INCLUDE 'common.h'
```

```
POTT=0.0d0
```

```
QINT=0.0d0
```

```
ent_loop: DO I=1,NAN
```

```
    POTT=POTT+POT(I)
```

```
    QINT=QINT+QIN(I)
```

```
END DO ent_loop
```

```
QINT=QINT*ENUNIT
```

```
POTT=POTT*ENUNIT
```

```
ENT=QINT+POTT
```

```
IF(KBOUND.EQ.0.OR.NRIGID.EQ.0) Then
```

```
    TEMPTR=QINT*2.0d0/BK/NDIM/real(NAN)
```

```
ElseIF (NRIGID.GE.NAN) Then
```

```
    TEMPTR=0.0d0
```

```
Else
```

```
    TEMPTR=QINT*2.0d0/BK/NDIM/real(NAN-NRIGID)
```

```
END IF
```

```
RETURN
```

```
END
```

## Listing of MSE627-MD code, Heating.f

```
! Heating of the material in a less violent manner then
! just by velocities distribution (as in Vel.f [KFLAG=2])

SUBROUTINE HEATING(TEMPTR,HDONE)
INCLUDE 'common.h'
Parameter (SKF=0.001d0)
IF(TEMPTR.GE.QTEM) THEN
    HDONE=1
    RETURN
ELSE
    SC=SQRT(1.0d0+SKF*(QTEM-TEMPTR)/(QTEM))
    t_loop: DO I=1,NAN
!         Sometimes you may want to heat up only a part of your system
!         IF(XD(3,I).GT.(ZCENTR+ZL/4.0d0)) CYCLE t_loop
!         IF(KHIST(J).NE.2) CYCLE t_loop
        Q1D(1,I)=Q1D(1,I)*SC
        Q1D(2,I)=Q1D(2,I)*SC

        IF(NDIM.EQ.3) Q1D(3,I)=Q1D(3,I)*SC
    End do t_loop
ENDIF

RETURN
END
```

## Listing of MSE627-MD code, Gather.f

- ! Gathering molecules to the computational cell
- ! It is necessary to gather particles back to the computational cell from time to time
- ! for systems with active diffusion. We have to avoid the situation when the particle
- ! diffuses farther than the size of the computational cell.

```
SUBROUTINE Gather ()
INCLUDE 'common.h'
the_loop: DO I=1,NAN
    I3=I+I+I
    IF (LIDX.EQ.1) THEN
        DX=X(I3-2)-XCENTR
        IF (DX.GT.XLHALF) X(I3-2)=X(I3-2)-XL
        IF (DX.LT.-XLHALF) X(I3-2)=X(I3-2)+XL
    ENDIF
    IF (LIDZ.EQ.1) THEN
        DZ=X(I3)-ZCENTR
        IF (DZ.GT.ZLHALF) X(I3)=X(I3)-ZL
        IF (DZ.LT.-ZLHALF) X(I3)=X(I3)+ZL
    ENDIF
    IF (LIDY.EQ.1) THEN
        DY=X(I3-1)-YCENTR
        IF (DY.GT.YLHALF) X(I3-1)=X(I3-1)-YL
        IF (DY.LT.-YLHALF) X(I3-1)=X(I3-1)+YL
    ENDIF
END DO the_loop
RETURN
END
```

```
! Write output files for further analysis
SUBROUTINE Swrite()
INCLUDE 'common.h'
DIMENSION TEMPA(LPMX)
CHARACTER*5 chtime
! Transferring velocities to A/ps, Energies to eV
Q1D(1:3,1:NAN)=Q1D(1:3,1:NAN)/DELTA
POT(1:NAN)=POT(1:NAN)*ENUNIT
QIN(1:NAN)=QIN(1:NAN)*ENUNIT
EN(1:NAN)=POT(1:NAN)+QIN(1:NAN)
TEMPA(1:NAN)=QIN(1:NAN)*2.0d0/BK/NDIM

ITIME=ANINT(TIME)
IF (ITIME.LT.1000) Then
    WRITE(chtime, FMT='(I3.3)') ITIME
ELSEIF (ITIME.LT.10000) Then
    WRITE(chtime, FMT='(I4.4)') ITIME
ELSEIF (ITIME.LT.100000) Then
    WRITE(chtime, FMT='(I5.5)') ITIME
ELSE
    PRINT *, "Simulation is too long... Time=",Time," ps."
    STOP
ENDIF

OPEN(UNIT=969,FILE='.///DRNAME(1:LDR)//time//trim(chtime)//.d')
```

```
REWIND 969
```

```
WRITE(969,166)(J,KHIST(J),XD(1,J),XD(2,J),XD(3,J), &
```

```
    Q1D(1,J),Q1D(2,J),Q1D(3,J),POT(J),QIN(J),TEMPA(J),EN(J),J=1,NAN)
```

```
166 FORMAT(I6,1X,I2,1X,E12.5,1X,E12.5,1X,E12.5,1X, &
```

```
    E10.3,1X,E10.3,1X,E10.3,1X,E10.3,1X,E10.3,1X,E10.3)
```

```
CLOSE (UNIT = 969)
```

```
!   Going back to the units used inside the MD loop
```

```
Q1D(1:3,1:NAN)=Q1D(1:3,1:NAN)*DELTA
```

```
POT(1:NAN)=POT(1:NAN)/ENUNIT
```

```
QIN(1:NAN)=QIN(1:NAN)/ENUNIT
```

```
RETURN
```

```
END
```

## Listing of MSE627-MD code, SetEnd.f

```
! THIS SUBROUTINE CLEANS UP AFTER THE MD LOOP

SUBROUTINE SetEnd()
INCLUDE 'common.h'

! Since we multiplied velocities by timestep DELTA at the beginning of the run
! (in SetInit.f) we have to divide by DELTA here to get real velocities
DO I=1,NAN3
    Q1(I)=Q1(I)/DELTA
ENDDO

RETURN
END
```

## Listing of MSE627-MD code, WriteEnd.f

```
! THIS SUBROUTINE WRITES DATA AT THE END OF THE RUN
SUBROUTINE WriteEnd()
INCLUDE 'common.h'

! Write output coordinate file
REWIND 16
WRITE(16,*) NAN,TIME
WRITE(16,*) XL,YL,ZL,XCENTR,YCENTR,ZCENTR
WRITE(16,*) (KTYPE(J),J=1,NAN)
WRITE(16,*) (KHIST(J),XD(1,J),XD(2,J),XD(3,J),J=1,NAN)
WRITE(16,*) (Q1D(1,J),Q1D(2,J),Q1D(3,J),J=1,NAN)

! Write file for plotting the final configuration
IF(MOV.EQ.1) THEN
    REWIND 18
    WRITE(18,505) (XD(1,J),XD(2,J),XD(3,J),J=1,NAN)
505    FORMAT(3(F11.3,1x))
ENDIF

! Write random numbers
REWIND 13
CALL Random_seed(SIZE = Kseed)
CALL Random_seed(GET = iseed)
WRITE(13,*) (ISEED(i), i=1,Kseed) ! for random_number(rww)
WRITE(13,*) U1,U2                ! for rn()

! Close the files
CLOSE (UNIT = 8)
CLOSE (UNIT = 9)
... Close other files ....

RETURN
END
```