Supplementary material for the work: "Path Integral Monte Carlo on a Sphere"

Riccardo Fantoni*
Università di Trieste, Dipartimento di Fisica, strada Costiera 11, 34151 Grignano (Trieste), Italy
(Dated: March 10, 2026)

Supplementary material to the article "Path Integral Monte Carlo on a Sphere".

Keywords: Path Integral; Monte Carlo; Sphere; Quantum Fluid; Bosons; Fermions; Anyons; Thermodynamics; Structure; Superfluidity; Sign Problem

I. INTRODUCTION

We here report the FORTRAN code listing of the code used in the simulations developed in the publication "Path Integral Monte Carlo on a Sphere". This is shown in Appendix A. The model studied in that publication was that of a many body fluid of N particles at $\boldsymbol{r}_i$, $i = 1, 2, \ldots, N$, on a sphere of radius a. Where $\boldsymbol{r} = (\theta, \varphi)$ with $\theta \in]-\pi/2, \pi/2]$ the polar angle and $\varphi \in]-\pi, \pi]$ the azimuthal angle. The metric of the sphere is $ds^2 = g_{\mu\nu} dr^\mu dr^\nu$ with a diagonal metric tensor $g_{\mu\nu}$ such that $g_{11} = a^2$ and $g_{22} = a^2 \cos^2\theta$.

The plane waves [1] Path Integral Monte Carlo (PIMC) method used in our computer experiments is the one described in Ref. [2]. A single MC step is made of the exchange of a randomly chosen pair of particles through the *Lévy construction* of two Brownian bridges between two randomly chosen timeslices on different particles paths (see Section V.G of Ref. [2] and references therein for the multislice move) and of a displacement of all the timeslices of a particle path chosen at random (the singleslice move). The exchange move allows to sample the permutation sum necessary in the calculation of the symmetric Bose-Einstein density matrix. The extent of the singleslice displacement is chosen at the beginning of the run through the input variable `ALPHA` or it can be adjusted during the run every `IRATIO` MC steps so to have acceptance ratios closest to 1/2 (this functionality is commented out in the present listing).

We discretize the imaginary time between 0 and `BETA` into `FTN0` timeslices of length τ=`LS`=`BETA`/`FTN0`. One usually wants to compare simulations at fixed `LS`. The multislice move requires the reconstruction of a randomly chosen number `MBM`≤`MBMM`≤`FNT0` of timeslices between a randomly chosen timeslice `CP` on a randomly chosen particle `IP` and a timeslice on a randomly chosen particle `KP`. Here `MBMM` is the maximum number of timeslices in a Brownian bridge. The singleslice move requires the displacement of all the `FTN0` timeslices of the path of the particle `IP` chosen at random. So a single MC `STEP` requires a maximum of 2`MBMM`+`FTN0` timeslice moves. In the code we wrote the FORTRAN instructions for the multislice move in lowercase and the ones for the singleslice move in uppercase. The simulation sets `KP`=`IP` in the excahnge move if the logical variable `IFB` is set to `FALSE`. So that if it is set to `TRUE` Bose-Einstein statistics is used, otherwise is used Boltzmann statistics. The simulation uses only the multislice move if the logical variable `IFBRIDGE` is `TRUE` and only the singleslice move if the logical variable `IFDISP` is `TRUE`. The simulation lasts for a total number of MC steps `NSTEP`. All this variables can be chosen in the `data-gr-rs.in` simulation configuration input file at startup.

On a sphere it is necessary to choose `DIM`= 2 with a number of particles N=`NP` of mass `FTM` with natural units $\hbar = k_B = 1$. One has to specify also the radius of the sphere `RADIUS` and the absolute temperature `TEMP`.

The pair potential between the particles `POTK` is encoded in `SUBROUTINE POT` at the end of the code listing. It includes the Coulomb potential that can be chosen by setting the string `POTK` equal to `COULOMB`. The ideal gas is obtained choosing the string `FREE`. This can be set up in the `data-gr-rs.in` simulation configuration input file. The variables `RCUT`, `SIG`, `EPSR`, `EPSA`, and `EPS` in the simulation configuration input file are used to specify other properties of a non Coulomb pair potential.

The simulation calculates both thermodynamic quantities and static structural properties. For the thermodynamic quantities we measure: the kinetic energy `KE`, the physical potential energy `V`, the geometric effective potential energy `W`= $-\sum_{i=1}^{N} \ln \sqrt{g(\boldsymbol{r}_i)}/\tau$ where g is the determinant of the sphere metric tensor, and the superfluid fraction `SF`. These quantities are stored in the output file `fort.9` every `IPRINT` MC steps. Whereas the path configurations output file `CNFILE` is stored in a *.xyz* format every `ISAVE` MC steps. The first `IEQUI` MC steps are used for the equilibration of the Markov chain. Quantities are only calculated after these equilibration steps. For the structural properties we measure the radial distribution function $g(r)$, with r the Euclidean distance, which is written in the output file `rdf.dat`. In the current implementation of the code, the $g(r)$ is only calculated during the displacement move every `ICALCG` timeslices.

If the variable `INIT` is set to 0 the simulation starts from a path with all particles at all timeslices on the equator $\theta = 0$ at $\varphi = 0$, if it is set to 1 it starts from a path with the particles randomly distributed uniformly on the equator with all timeslices copied above the first

* riccardo.fantoni@scuola.istruzione.it
[1] For a coherent states PIMC see Ref. [1].

one for each particle, if it is set to 2 it reads the paths configuration from the file `CNFILE` stored on disk, if it is set to 3 it does a complete restart from the file `CNFILE` and the frozen configuration `back.dat` both stored on disk.

The variable `SEED`, set in the simulation configuration input file, is the positive integer seed of the pseudo random number generator.

For the RPIMC algorithm also used in the work for the simulations of fermions [3, 4] and anyons we modified the code presented here as described in the main text of the article.

ACKNOWLEDGMENTS

I would like to thank prof. Saverio Moroni for his support in the development of the brownian bridge and the consequent particles permutation sampling.

Appendix A: The code

This is the code used for the PIMC computer experiment. We list here the main FORTRAN code `pimc-sph-rs.f` with the included `mc-sph.par` parameters file. And the input simulation configuration file `data-gr-rs.in` that is read at the beginning of the run.

```
*******************************************************************************
*** pimc-sph-rs.f *************************************************************
*******************************************************************************

      PROGRAM PIMC
      IMPLICIT NONE
C     ****************************************************************
C     ** MONTE CARLO SIMULATION PROGRAM FOR [PW] PATH INTEGRAL       **
C     **                                                            **
C     ** on a sphere of radius RADIUS                               **
C     **                                                            **
C     ** space dimensions:    DIM=2                                 **
C     ** number of particles: NP                                    **
C     ** number of timesteps: FTNO                                  **
C     ** units:               hbar=k_B=1                            **
C     ** BETA=1/TEMP=FTNO*LS  (LS imaginary time spacing)           **
C     **                                                            **
C     ** OBSERVABLES:                                               **
C     ** kinetic energy       KE                                    **
C     ** potential energy     V                                     **
C     ** forces virial        W                                     **
C     ** superfluid fraction  SF                                    **
C     ** radial distribution function on 'rdf.dat'                  **
C     ****************************************************************
      INCLUDE       'mc-sph.par'                        ! parameters

      INTEGER*4     SEED
      INTEGER*8     IDIM, STEP, I, J, K
      INTEGER*8     C1, CP, IP, KP, NIP, NKP, LL, OUT1, OUT2, PIP
      INTEGER*8     INIT, NSTEP, NS, IPRINT, ISAVE, IRATIO, IEQUI
      INTEGER*8     MBMM, MCM, PREV(MNP)
      INTEGER*8     CYC(MNP), NSWAP

      REAL*8        DENS, TEMP, BETA, DENSLJ
      REAL*8        DRMAX, ALPHA, CDIM
      REAL*8        RANF, DUMMY, SR9, SR3
      REAL*8        VLRC, VLRC6, VLRC12, WLRC, WLRC6, WLRC12
      REAL*8        ACM, ACATMA, ACATMAS, ACATMAB
      REAL*8        V, VNEW, VOLD, VEND, DELTV, VN, VS
      REAL*8        W, WNEW, WOLD, WEND, DELTW, WN, WS
      REAL*8        KE, KENEW, KEOLD, DELTKE, KEEND
      REAL*8        AVV, ACV, ACVSQ, FLV
      REAL*8        AVW, ACW, ACWSQ, FLW
      REAL*8        AVKE, ACKE, ACKESQ, FLKE
      REAL*8        ACT, ACTNEW, ACTOLD, DELTACT, DELTACTB
      REAL*8        RXIOLD(MDIM), RXINEW(MDIM)
      REAL*8        RXP(MDIM,0:N), RXPP(MDIM,0:N)
      REAL*8        STD, PS, KKK, VVV, WWW
      REAL*8        RATIO, RATIOS, RATIOB
      REAL*8        WIND, ACWIND, SFI, SF, IC
      REAL*8        GA

      INTEGER*8     BIN, MAXBIN, MAXBIN2, HIST(1000000), ICALCG
      REAL*8        CONST, RLOWER, RUPPER
      REAL*8        TSTEP, NIDEAL, GR

      CHARACTER     CNFILE*30, POTK*10

      LOGICAL       OVRLAP, IFB, IFDISP, IFBRIDGE

C     PI = ACOS(-1.d0)
      OPEN(UNIT = 16, FILE = 'back.dat', STATUS = 'UNKNOWN')
      OPEN(UNIT = 15, FILE = 'rdf.dat' , STATUS = 'UNKNOWN')

C     ****************************************************************
C     ** READ INPUT DATA                                            **
C     ****************************************************************
      WRITE(*,'('' ********** PROGRAM PIMC ************      ''/)')
      WRITE(*,'('' PLANE WAVES                                    '')')
      WRITE(*,'('' PATH INTEGRAL MONTE CARLO PROGRAM          '')')
      OPEN (UNIT=10, FILE='data-gr-rs.in', STATUS='UNKNOWN')
      READ (10,*) I
      READ (10,*) DIM
      READ (10,*) I
      READ (10,*) SEED
      READ (10,*) I
      READ (10,*) IFB
      READ (10,*) I
      READ (10,*) NP
      READ (10,*) I
      READ (10,'(A)') POTK
      READ (10,*) I
      READ (10,*) FTNO
      READ (10,*) I
      READ (10,*) FTM
      READ (10,*) I
      READ (10,*) NSTEP
      READ (10,*) I
      READ (10,*) IPRINT
      READ (10,*) I
      READ (10,*) ISAVE
      READ (10,*) I
      READ (10,*) IEQUI
      READ (10,*) I
      READ (10,*) IRATIO
      READ (10,*) I
      READ (10,*) ICALCG
      READ (10,*) I
      READ (10,'(A)') CNFILE
      READ (10,*) I
      READ (10,*) INIT
      READ (10,*) I
      READ (10,*) RADIUS
      READ (10,*) I
      READ (10,*) TEMP
      READ (10,*) I
      READ (10,*) ALPHA
      READ (10,*) I
      READ (10,*) MBMM
      READ (10,*) I
      READ (10,*) RCUT
      READ (10,*) I
      READ (10,*) SIG
      READ (10,*) I
      READ (10,*) EPSR
      READ (10,*) I
      READ (10,*) EPSA
      READ (10,*) I
      READ (10,*) EPS
      READ (10,*) I
      READ (10,*) IFDISP
      READ (10,*) I
      READ (10,*) IFBRIDGE
      CLOSE (UNIT=10)
      WRITE(*,'('' IN DIMENSION (1,2,3,...)''I12/)') DIM
      WRITE(*,'('' ***********************************      ''/)')

      GOTO 1111                    ! comment if input from keyboard

      WRITE(*,'('' ********** PROGRAM PIMC ************      ''/)')
      WRITE(*,'('' PLANE WAVES                                    '')')
      WRITE(*,'('' PATH INTEGRAL MONTE CARLO PROGRAM          '')')

      WRITE(*,'('' ENTER THE SPATIAL DIMENSIONS               '')')
      READ (*,*) DIM
      WRITE(*,'('' ENTER SEED FOR RANDOM SEQUENCE             '')')
      READ (*,*) SEED
      WRITE(*,'('' IF BOSE (T/F)                              '')')
      READ (*,*) IFB
      WRITE(*,'('' ENTER THE NUMBER OF PARTICLES < MNP        '')')
      READ (*,*) NP
      WRITE(*,'('' ENTER TYPE OF PAIR POTENTIAL (Many Body)   '')')
      READ (*,'(A)') POTK
      WRITE(*,'('' ENTER THE NUMBER OF DISCRETIZATIONS < N    '')')
      READ (*,*) FTNO
      WRITE(*,'('' ENTER THE BARE MASS                        '')')
      READ (*,*) FTM
      WRITE(*,'('' ENTER NUMBER OF CYCLES                     '')')
      READ (*,*) NSTEP
      WRITE(*,'('' ENTER NUMBER OF STEPS BETWEEN OUTPUT LINES '')')
      READ (*,*) IPRINT
      WRITE(*,'('' ENTER NUMBER OF STEPS BETWEEN DATA SAVES   '')')
      READ (*,*) ISAVE
      WRITE(*,'('' ENTER NUMBER OF STEPS FOR EQUILIBRATION    '')')
      READ (*,*) IEQUI
      WRITE(*,'('' ENTER INTERVAL FOR UPDATE OF MAX. DISPL.   '')')
      READ (*,*) IRATIO
      WRITE(*,'('' ENTER INTERVAL FOR RDF CALCULATION         '')')
      READ (*,*) ICALCG
      WRITE(*,'('' ENTER THE CONFIGURATION FILE NAME          '')')
      READ (*,'(A)') CNFILE
      WRITE(*,'('' ENTER 0 INIT Z, 1 INIT, 2 REST C, 2 REST F '')')
      READ (*,*) INIT
      WRITE(*,'('' ENTER THE SPHERE RADIUS                    '')')
      READ (*,*) RADIUS
      WRITE(*,'('' ENTER THE TEMPERATURE                      '')')
```

```
      READ (*,*) TEMP
      WRITE(*,'('' ENTER MAX. DISPLACEMENT DRMAX/SIGMA        '')')
      READ (*,*) ALPHA
      WRITE(*,'('' ENTER MAXIMUM NUMBER OF BRIDGE TIMESLICES  '')')
      READ (*,*) MBMM
      WRITE(*,'('' ENTER THE POTENTIAL CUTOFF DISTANCE (~ 2.5)'')')
      READ (*,*) RCUT
      WRITE(*,'('' ENTER SIG                                  '')')
      READ (*,*) SIG
      WRITE(*,'('' ENTER EPSR                                 '')')
      READ (*,*) EPSR
      WRITE(*,'('' ENTER EPSA                                 '')')
      READ (*,*) EPSA
      WRITE(*,'('' ENTER EPS                                  '')')
      READ (*,*) EPS
      WRITE(*,'('' IF DISPLACEMENT MOVE (T/F)                 '')')
      READ (*,*) IFDISP
      WRITE(*,'('' IF BRIDGE MOVE (T/F)                       '')')
      READ (*,*) IFBRIDGE
      WRITE(*,'('' IN DIMENSION (1,2,3,...)''I12/)') DIM
      WRITE(*,'('' ************************************      ''/)')

 1111 CONTINUE

      IF (MBMM .GT. FTN0) THEN
         WRITE(*,*) "number of timeslices in bridge > ",FTN0
         STOP
      ENDIF

      IF (DIM .NE. 2) THEN
         WRITE(*,*) "DIM not equal to 2"
         STOP
      ENDIF

C     ** INITIALIZE RANDOM NUMBER GENERATOR **

      CALL SRAND ( SEED )
      IF (SEED.EQ.1) THEN
         CALL SRAND ( 0 )
      ENDIF

C     ** DENSITY **

      DENS = NP/4.d0/PI/RADIUS**2

C     ** INVERSE TEMPERATURE BETA **

      BETA = (1.d0/TEMP)

C     ** IMAGINARY TIMESTEP **

      LS  = BETA/FTN0      ! time step
      STD = (LS/FTM)**0.5  ! standatd deviation of free-particle

C     ** MOMENT OF INERTIA OF A UNITARY SPHERICAL SHELL **

      IC = 2*(NP*FTM)/3

C    ** WRITE INPUT DATA **

      WRITE(*,'('' SEED                     '',I10   )') SEED
      WRITE(*,'('' POTENTIAL                '',A     )') POTK
      WRITE(*,'('' DIMENSIONS               '',I10   )') DIM
      WRITE(*,'('' NUMBER OF PARTICLES      '',I10   )') NP
      WRITE(*,'('' NUMBER OF CYCLES         '',I10   )') NSTEP
      WRITE(*,'('' NUMBER OF EQUIL. STEPS   '',I10   )') IEQUI
      WRITE(*,'('' OUTPUT FREQUENCY         '',I10   )') IPRINT
      WRITE(*,'('' SAVE FREQUENCY           '',I10   )') ISAVE
      WRITE(*,'('' RATIO UPDATE FREQUENCY   '',I10   )') IRATIO
      WRITE(*,'('' CONFIGURATION FILE  NAME '',A     )') CNFILE
      WRITE(*,'('' TEMPERATURE              '',E10.4 )') TEMP
      WRITE(*,'('' DENSITY                  '',E10.4 )') DENS
      WRITE(*,'('' PARTICLE MASS            '',E10.4 )') FTM
      WRITE(*,'('' # TIME SLICES            '',I10   )') FTN0
      WRITE(*,'('' TIME STEP                '',E10.4 )') LS

C    ** CONVERT INPUT DATA TO PROGRAM UNITS **

C       SIGMA  = ( DBLE ( NP ) / DENS ) ** ( 1.0 / DIM )
      SIGMA   = 2*PI
      SIGM(1) = PI
      SIGM(2) = 2*PI
      DRMAX   = ALPHA * SIGMA

      IF (POTK .EQ. 'LJ') THEN
         DENSLJ = DENS * SIG ** DBLE( DIM )
C         RCUT = SIGMA/2.d0/SIG
      ENDIF

      MAXBIN  = INT ( 2*RADIUS/DELR )     ! Euclidean
C       MAXBIN  = INT (PI*RADIUS/DELR )     ! Geodesic
      MAXBIN2 = INT ( MAXBIN/2.0 )
      DO I = 1, MAXBIN
         HIST(I) = 0
      ENDDO

      IF (DIM .eq. 3) CONST = 4.d0 * PI * DENS / 3.d0
      IF (DIM .eq. 2) CONST = PI * DENS
      IF (DIM .eq. 1) CONST = 2.d0 * DENS

C     ** INITIALIZE CONFIGURATION **

      DO I = 1, NP
         NEXT(I) = I
         LEXT(I) = I
         DO J = 1, FTN0
            MEXT(J,I)  = I
         ENDDO
      ENDDO

C     ** ZERO ACCUMULATORS **

      ACV     = 0.0
      ACVSQ   = 0.0
      FLV     = 0.0
      ACW     = 0.0
      ACWSQ   = 0.0
      FLW     = 0.0
      ACKE    = 0.0
      ACKESQ  = 0.0
      FLKE    = 0.0
      ACWIND  = 0.0
      ACM     = 0.0
      ACATMA  = 0.0
      ACATMAB = 0.0
      ACATMAS = 0.0

      IF ( INIT .EQ. 0 ) THEN
         PRINT*,"PATHS INITIALIZATION ..............."
         RX=0.                ! all paths zeroed
         NS = NSTEP
         OPEN (UNIT = 9, STATUS ='UNKNOWN')
      ELSEIF ( INIT .EQ. 1) THEN
         PRINT*,"PATHS INITIALIZATION ..............."
C     ** READ INITIAL CONFIGURATION **
         CALL INITCNSPH ( CNFILE ) ! random configuration on equator
         CALL READCN ( CNFILE )
         NS = NSTEP
         OPEN (UNIT = 9, STATUS ='UNKNOWN')
      ELSEIF ( INIT .EQ. 2) THEN
         PRINT*,"RESTART FROM CONFIG  ..............."
C     ** READ INITIAL CONFIGURATION **
         CALL READCN ( CNFILE )
         NS = NSTEP
         OPEN (UNIT = 9, STATUS ='OLD', ACCESS ='APPEND')
      ELSEIF ( INIT .EQ. 3) THEN
         PRINT*,"RESTART FULL         ..............."
         CALL READCN ( CNFILE )
         READ(16,*)(HIST(I),I=1,MAXBIN),
     &          ACV,ACVSQ,ACW,ACWSQ,ACKE,ACKESQ,ACWIND,ACM,NS
         NS=NS+NSTEP
         OPEN (UNIT = 9, STATUS ='OLD', ACCESS ='APPEND')
      ELSE
         WRITE (*,*) 'INIT different from 0,1,2,3'
         STOP
      ENDIF

C     ** CALCULATE LONG RANGE CORRECTIONS    **
C     ** SPECIFIC TO THE LENNARD JONES FLUID **
      IF (DIM .EQ. 1) CDIM = 1
      IF (DIM .EQ. 2) CDIM = PI
      IF (DIM .EQ. 3) CDIM = 2*PI

      SR3 = - RCUT ** (- 6. + DIM)/(-6. + DIM)
      SR9 = - RCUT ** (- 12. + DIM)/(-12. + DIM)

      VLRC12 =   4 * EPS * CDIM * DENSLJ * NP * SR9
      VLRC6  = - 4 * EPS * CDIM * DENSLJ * NP * SR3
      VLRC   =   VLRC12 + VLRC6
      WLRC12 =   4.0  * VLRC12
      WLRC6  =   2.0  * VLRC6
      WLRC   =   WLRC12 + WLRC6

C     ** WRITE OUT SOME USEFUL INFORMATION **

      WRITE(*,'('' SIGMA                    '',E10.4)') SIGMA
      WRITE(*,'('' MAXIMUM DISPLACEMENT     '',E10.4)') DRMAX

C    ** CALCULATE INITIAL ENERGY AND CHECK FOR OVERLAPS **

      CALL SUMUP (POTK, OVRLAP, KE, V, W)

      IF (POTK .EQ. 'LJ') THEN
         VS = ( V + VLRC )
         WS = ( W + WLRC )
      ELSE
         VS = V
         WS = W
      ENDIF

      WRITE(*,'('' INITIAL V                '',E10.4)') VS
      WRITE(*,'('' INITIAL W                '',E10.4)') WS
      WRITE(*,'('' INITIAL KE               '',E10.4)') KE

      WRITE(*,'(//'' START OF MARKOV CHAIN              ''//)')
      WRITE(*,'('' NMOVE     RATIO       ACTION          ''/)')

C     ***************************************************************
C     ** LOOPS OVER ALL CYCLES AND ALL TIME SLICES                  **
C     ***************************************************************

      DO 100 STEP = NS-NSTEP+1, NS

         CP = INT(FTN0*RANF(DUMMY))+1    ! select a random timeslice
         MBM = INT((MBMM-1)*RANF(DUMMY))+2  ! # timeslices in bridge
         IP = INT(NP*RANF(DUMMY))+1      ! select a particle
         KP = INT(NP*RANF(DUMMY))+1      ! select another particle
         IF (.NOT.IFB) KP = IP           ! for boltzmann statistics
         IF (IFDISP) GOTO 77   ! uncomment if only displacement move

C     ***************************************************************
C     ** BRIDGE&SWAP MOVE (BOSE STATISTICS)                         **
C     ***************************************************************

C     direct mapping from sphere rx to plane rp
          call mappingd()
          call equiv(rpn,rp)

          mcm = cp+mbm-floor((cp+mbm-.1)/ftn0)*ftn0

          nip=next(ip)
          nkp=next(kp)
```

```
            vold=0.d0
            wold=0.d0
            if(cp+mbm.le.ftn0)then
               do i=cp+1,mcm-1
                  call pppnergy ( potk, rx(:,i,ip), ip,
     :                 rx(:,i,kp), kp, i,
     :                 vvv, www )
                  vold=vold+vvv
                  wold=wold+www
               enddo
            else
               do i=cp+1,ftn0
                  call pppnergy ( potk, rx(:,i,ip), ip,
     :                 rx(:,i,kp), kp, i,
     :                 vvv, www )
                  vold=vold+vvv
                  wold=wold+www
               enddo
               do i=1,mcm-1
                  call pppnergy ( potk, rx(:,i,nip), nip,
     :                 rx(:,i,nkp), nkp, i,
     :                 vvv, www )
                  vold=vold+vvv
                  wold=wold+www
               enddo
            endif

            keold=0.d0
            do k=1,np
               if(k.eq.ip.or.k.eq.kp.or.k.eq.nip.or.k.eq.nkp)then
                  do i=1,ftn0
                     call kknergy ( rx(:,i,k), k, i,
     :                    kkk )
                     keold=keold+kkk
                  enddo
               endif
            enddo

c     bridge move
            if(cp+mbm.le.ftn0)then
               call bridge(rp(:,cp,ip),rp(:,mcm,kp),std,
     :              rxp,out1)
               if (ip.ne.kp) then
                  call bridge(rp(:,cp,kp),rp(:,mcm,ip),std,
     :                 rxpp,out2)
               endif
            else
               call bridge(rp(:,cp,ip),rp(:,mcm,nkp),std,
     :              rxp,out1)
               if (ip.ne.kp) then
                  call bridge(rp(:,cp,kp),rp(:,mcm,nip),std,
     :                 rxpp,out2)
               endif
            endif

c     mapping rxp and rxpp from plane to sphere

            vnew=0.d0
            wnew=0.d0
            ll=0
            if(cp+mbm.le.ftn0)then
               do i=cp+1,mcm-1
                  ll=ll+1
                  if (ip.eq.kp) then
                     call ppnergy ( potk, rxp(:,ll), ip,
     :                    rxp(:,ll), kp, i,
     :                    vvv, www )
                  else
                     call ppnergy ( potk, rxp(:,ll), ip,
     :                    rxpp(:,ll), kp, i,
     :                    vvv, www )
                  endif
                  vnew=vnew+vvv
                  wnew=wnew+www
               enddo
            else
               do i=cp+1,ftn0
                  ll=ll+1
                  if (ip.eq.kp) then
                     call ppnergy ( potk, rxp(:,ll), ip,
     :                    rxp(:,ll), kp, i,
     :                    vvv, www )
                  else
                     call ppnergy ( potk, rxp(:,ll), ip,
     :                    rxpp(:,ll), kp, i,
     :                    vvv, www )
                  endif
                  vnew=vnew+vvv
                  wnew=wnew+www
               enddo
               do i=1,mcm-1
                  ll=ll+1
                  if (ip.eq.kp) then
                     call ppnergy ( potk, rxp(:,ll), nip,
     :                    rxp(:,ll), nkp, i,
     :                    vvv, www )
                  else
                     call ppnergy ( potk, rxpp(:,ll), nip,
     :                    rxp(:,ll), nkp, i,
     :                    vvv, www )
                  endif
                  vnew=vnew+vvv
                  wnew=wnew+www
               enddo
            endif
            deltv=vnew-vold

c     execute the bridge move
            acatmab = acatmab + 1.d0
            call updaten(cp,rxp,ip,nkp)
            if (ip.ne.kp) then
               call updaten(cp,rxpp,kp,nip)
               call swapn(cp,ip,nip,kp,nkp)
               call switch(next(ip),next(kp))
               call switch(mext(cp,ip),mext(cp,kp))
               acatmas = acatmas + 1.d0
            endif

c     calculate the ratio of transition probabilities
            call gauss(cp,ga)

c     inverse mapping
            call equiv(rp,rpn)
            call equiv(rpn,rx)
            call mappingi()

            kenew=0.d0
            do k=1,np
               if(k.eq.ip.or.k.eq.kp.or.k.eq.nip.or.k.eq.nkp)then
                  do i=1,ftn0
                     call kknergy ( rx(:,i,k), k, i,
     :                    kkk )
                     kenew=kenew+kkk
                  enddo
               endif
            enddo
            deltke=kenew-keold

c     calculate the acceptance probability
            call acc_p(deltv,deltke,ps)
            ps=ps*ga

c     accept/reject
            if (ps.gt.ranf(dummy)) then
               ke=ke+deltke
               v=v+vnew-vold
               w=w+wnew-wold
            else
               call equiv(rx,rpn)
               acatmab = acatmab - 1.d0
               if (ip.ne.kp) then
                  call switch(next(ip),next(kp))
                  call switch(mext(cp,ip),mext(cp,kp))
                  acatmas = acatmas - 1.d0
               endif
            endif

c     build the permutations cycles in lext
            call permcyc()
c     count number of swaps of two particles
            call cycles(cyc,nswap)

c     balance ke between bridge and displace
c           call sumupke ( ke )

            IF (STEP.GT.IEQUI) THEN
               ACM = ACM + 1.0

C     ** CALCULATE INSTANTANEOUS VALUES **

               IF (POTK .EQ. 'LJ') THEN
                  VN = ( V + VLRC )
               ELSE
                  VN = V - W
                  WN = W
               ENDIF
               CALL CWIND( WIND )
               IF (WIND.GT.3*BETA*IC*2/FTM**2) WIND=3*BETA*IC*2/FTM**2

C     ** ACCUMULATE AVERAGES **

               ACV    = ACV    + VN
               ACVSQ  = ACVSQ  + VN*VN
               ACW    = ACW    + WN
               ACWSQ  = ACWSQ  + WN*WN
               ACKE   = ACKE   + KE
               ACKESQ = ACKESQ + KE*KE
               ACWIND = ACWIND + WIND
            ENDIF

            IF (IFBRIDGE) GOTO 97

C     ****************************************************************
C     ** ENDS BRIDGE&SWAP MOVE                                      **
C     ****************************************************************

 77         CONTINUE

C     ****************************************************************
C     ** DISPLACEMENT MOVE                                          **
C     ****************************************************************

C     ** PREVIOUS IP **

            DO I=1,NP
               J=NEXT(I)
               PREV(J)=I
            ENDDO

            NIP = NEXT(IP)
            PIP = PREV(IP)

            DO 90 C1 = 1, FTN0

               DO IDIM = 1, DIM
                  RXIOLD(IDIM) = RX(IDIM,C1,IP)
               ENDDO

C     ** CALCULATE THE ENERGY OF I IN THE OLD CONFIGURATION **

               CALL PENERGY ( POTK, RXIOLD, IP, C1,
     :              VOLD, WOLD )
```

```
              CALL KENERGY ( RXIOLD, IP, C1,
     :             KEOLD )

C     ** INSTANTANEOUS VALUE OF THE ACTION **

              ACTOLD = KEOLD + VOLD

C     ** MOVE I AND PICKUP THE CENTRAL IMAGE **

              DO IDIM = 1, DIM
                 RXINEW(IDIM) = RXIOLD(IDIM) +
     :                ( 2.0 * RANF ( DUMMY ) - 1.0 )*DRMAX
                 RXINEW(IDIM) = RXINEW(IDIM) -
     :                DNINT ( RXINEW(IDIM)/SIGM(IDIM) )*SIGM(IDIM)
              ENDDO

C     ** CALCULATE THE ENERGY OF I IN THE NEW CONFIGURATION **

              CALL PENERGY ( POTK, RXINEW, IP, C1,
     :             VNEW, WNEW )

              CALL KENERGY ( RXINEW, IP, C1,
     :             KENEW )

C     ** INSTANTANEOUS VALUE OF THE ACTION **

              ACTNEW = KENEW + VNEW

              DELTV  = VNEW  - VOLD
              DELTW  = WNEW  - WOLD
              DELTKE = KENEW - KEOLD

C     ** CHECK FOR ACCEPTANCE **

              DELTACT  = ACTNEW - ACTOLD
              DELTACTB = LS*DELTACT

              IF ( EXP ( - DELTACTB ) .GT. RANF ( DUMMY ) ) THEN
                 V          = V  + DELTV
                 W          = W  + DELTW
                 KE         = KE + DELTKE
                 ACATMA = ACATMA + 1.0

                 DO IDIM = 1, DIM
                    RX(IDIM,C1,IP)  = RXINEW(IDIM)
C     IMAGINARY TIME PERIODIC BOUNDARY CONDITIONS
                    IF (C1.EQ.1) THEN
                       RX(IDIM,FTNO+1,PIP)=RX(IDIM,C1,IP)
                    ENDIF
                    IF (C1.EQ.FTNO) THEN
                       RX(IDIM,0,NIP)=RX(IDIM,C1,IP)
                    ENDIF
                 ENDDO
              ENDIF

              IF (STEP.GT.IEQUI) THEN
                 ACM = ACM + 1.0

C     ** CALCULATE INSTANTANEOUS VALUES **

C     HISTOGRAM FOR g(r)
                 IF ( MOD ( C1, ICALCG ) .EQ. 0 ) THEN
                    CALL GOFR( C1, MAXBIN, HIST )
                 ENDIF

                 IF (POTK .EQ. 'LJ') THEN
                    VN = ( V + VLRC )
                 ELSE
                    VN = V - W
                    WN = W
                 ENDIF
                 CALL CWIND( WIND )
                 IF (WIND.GT.3*BETA*IC*2/FTM**2) WIND=3*BETA*IC*2/FTM**2

C     ** ACCUMULATE AVERAGES **

                 ACV    = ACV    + VN
                 ACVSQ  = ACVSQ  + VN*VN
                 ACW    = ACW    + WN
                 ACWSQ  = ACWSQ  + WN*WN
                 ACKE   = ACKE   + KE
                 ACKESQ = ACKESQ + KE*KE
                 ACWIND = ACWIND + WIND
              ENDIF

C     ******************************************************************
C     ** END DISPLACEMENT MOVE                                        **
C     ******************************************************************

 90        CONTINUE

C     ******************************************************************
C     ** ENDS LOOP OVER TIME SLICES                                   **
C     ******************************************************************

 97        CONTINUE

C     ** WRITE OUT THE INSTANTANEOUS VALUES ON FORT.9 **

           SFI = WIND  *FTM**2/BETA/IC/2
           SF  = ACWIND*FTM**2/BETA/IC/2/ACM

           CALL FLUSH(9)
           IF ( MOD ( STEP, IPRINT ) .EQ. 0 ) THEN
              WRITE(9,*) STEP, DIM*NP/2/LS-KE/FTNO, VN/FTNO, WN/FTNO, SFI
           ENDIF

C     ** CREATE POSITION DISTRIBUTION **

C          CALL DISTR(30_8,STEP*FTNO*NP/30)

C     ** PERFORM PERIODIC OPERATIONS **

           IF ( MOD ( STEP, IRATIO ) .EQ. 0 ) THEN

C     ** ADJUST MAXIMUM DISPLACEMENT **

              RATIO  = ACATMA  / DBLE ( FTNO * IRATIO )
              RATIOB = ACATMAB / DBLE ( IRATIO )
              RATIOS = ACATMAS / DBLE ( IRATIO )

              IF ( RATIO .GT. 0.5 ) THEN

C                 DRMAX  = DRMAX  * 1.05

              ELSE

C                 DRMAX  = DRMAX  * 0.95

              ENDIF

              ACATMA  = 0.0
              ACATMAB = 0.0
              ACATMAS = 0.0

           ENDIF

           IF ( MOD ( STEP, IPRINT ) .EQ. 0 ) THEN

C     ** WRITE OUT RUNTIME INFORMATION **

              WRITE(*,*) '          step            acm  dr/sig    ard
     :     arb       ars'
              WRITE(*,'(2(2XI13),(2Xf6.4),4(2Xf8.6))') STEP, INT(ACM),
     :                             ALPHA, RATIO, RATIOB, RATIOS
              WRITE(*,*) '            ke              v              w
     :          sf'
              WRITE(*,'(4(2XE13.7))') KE/FTNO, VN/FTNO, WN/FTNO,
     :                               SFI
              WRITE(*,*) '          <ke>            <v>            <w>
     :         <sf>'
              WRITE(*,'(4(2XE13.7))') ACKE/ACM/FTNO, ACV/ACM/FTNO,
     :                               ACW/ACM/FTNO, SF
              WRITE(*,*) '        nswap'
              WRITE(*,'(I3)') NSWAP
           ENDIF

           IF ( MOD ( STEP, ISAVE ) .EQ. 0 ) THEN

C     ** WRITE OUT THE CONFIGURATION AT INTERVALS **

              CALL WRITCN ( CNFILE )
              REWIND(16)
              CALL FLUSH(16)
              WRITE(16,*)(HIST(I),I=1,MAXBIN),
     &             ACV,ACVSQ,ACW,ACWSQ,ACKE,ACKESQ,ACWIND,ACM,STEP

C         ** WRITE OUT THE g(r) **

              IF ( STEP .GT. IEQUI ) THEN
                 TSTEP = DBLE((STEP - IEQUI)*INT(FTNO/ICALCG))
                 REWIND(15)
                 DO BIN = 1, MAXBIN
                    RLOWER = DBLE ( BIN - .5)*DELR
                    RUPPER = RLOWER + DELR
                    NIDEAL = (RUPPER/2/RADIUS)**DIM-
     :                   (RLOWER/2/RADIUS)**DIM
                    GR = DBLE(HIST(BIN))/TSTEP/DBLE(NP)**2/NIDEAL
                    CALL FLUSH(15)
                    WRITE (15,*) DBLE(BIN-1)*DELR, GR
                 ENDDO
              ENDIF

           ENDIF

 100    CONTINUE

C     ******************************************************************
C     ** ENDS THE LOOP OVER CYCLES                                    **
C     ******************************************************************

        WRITE(*,'(//'' END OF MARKOV CHAIN          ''//)')

C     ** CHECKS FINAL VALUE OF THE POTENTIAL ENERGY IS CONSISTENT **

        CALL SUMUP (POTK, OVRLAP, KEEND, VEND, WEND)

        IF ( ABS(VEND - V) .GT. 1.0d-03 ) THEN

           WRITE(*,'('' PROBLEM WITH V ENERGY !!!'')')
           WRITE(*,'('' VEND   = '', E20.6)') VEND
           WRITE(*,'('' V      = '', E20.6)') V

        ENDIF

        IF ( ABS(KEEND - KE) .GT. 1.0d-03 ) THEN

           WRITE(*,'('' PROBLEM WITH KE ENERGY !!!'')')
           WRITE(*,'('' KEEND  = '', E20.6)') KEEND
           WRITE(*,'('' KE     = '', E20.6)') KE

        ENDIF

C     ** WRITE OUT THE FINAL CONFIGURATION FROM THE RUN **

        CALL WRITCN ( CNFILE )

C     ** CALCULATE AND WRITE OUT RUNNING AVERAGES **

        AVV   = ACV / ACM
        ACVSQ = ( ACVSQ / ACM ) - AVV ** 2
```

```
      AVKE   = ACKE / ACM
      ACKESQ = ( ACKESQ / ACM ) - AVKE ** 2

C    ** CALCULATE FLUCTUATIONS **

      IF ( ACVSQ  .GT. 0.0 ) FLV  = SQRT ( ACVSQ/ACM )/FTN0
      IF ( ACKESQ .GT. 0.0 ) FLKE = SQRT ( ACKESQ/ACM )/FTN0

      WRITE(*,'(/'' AVERAGES ''/ )')
      WRITE(*,'('' <V/N>                = '',E12.6)') AVV/FTN0
      WRITE(*,'('' <KE/N>               = '',E12.6)') AVKE/FTN0

      WRITE(*,'(/'' FLUCTUATIONS ''/)')

      WRITE(*,'('' FLUCTUATION IN <V/N>  = '',E12.6)') FLV
      WRITE(*,'('' FLUCTUATION IN <KE/N> = '',E12.6)') FLKE
      WRITE(*,'(/'' END OF SIMULATION '')')

C     ******************************************************************
C     ******************************************************************
C     ******************************************************************
C     ** THE END                                                      **
C     ******************************************************************
C     ******************************************************************
C     ******************************************************************
      CLOSE(UNIT = 9 )
      CLOSE(UNIT = 15)
      CLOSE(UNIT = 16)
      STOP
      END


      subroutine permcyc()
      implicit none
!     given a particle i, lext(i) returns the cycle it belongs to
      INCLUDE       'mc-sph.par'

      integer*8 i,j,k,ii(mnp),jj,kk,ll

      kk=1
      do i=1,np
         ll=1
         ii(ll)=i
 1       jj=next(ii(ll))
         do j=1,ll
            if(jj.eq.ii(j))goto 2
         enddo
         ll=ll+1
         ii(ll)=jj
         goto 1
 2       do k=1,ll
            lext(ii(k))=kk
         enddo
         kk=kk+1
      enddo

      return
      end


      subroutine cycles(cyc,ns)
      implicit none
!     cyc(i) returns the number of particles in cycle of type i
!     ns returns the total number of 2 particles swaps
      INCLUDE       'mc-sph.par'
      integer*8 i,cyc(mnp),ns

      ns=0
      cyc=0
      do i=1,np
         cyc(lext(i))=cyc(lext(i))+1
      enddo
      do i=1,np
         if (cyc(i).ne.0) ns=ns+cyc(i)-1
      enddo

      return
      end


      subroutine switch(i,j)
!     switch i and j
      implicit none
      integer*8 i,j,k
      k=i
      i=j
      j=k
      return
      end


      subroutine acc_p(dv,dke,p)
      implicit none
!     complete acceptance probability
      INCLUDE       'mc-sph.par'

      real*8 p,dke,dv

      p=exp(-ls*dv)        ! contribution from the pair potential
      p=p*exp(-ls*dke)     ! contribution from the kinetic energy

      return
      end


      subroutine updaten(j,rxp,ip,nip)
      implicit none
!     updates a portion of the current path x using the proposed path xp
      INCLUDE       'mc-sph.par'

      integer*8 ip,nip,kp,nkp,j,l,k,idim
      integer*8 mcm
      real*8 rxp(mdim,0:n)

      mcm = j+mbm-floor((j+mbm-.1)/ftn0)*ftn0

      l=0
      if(j+mbm.le.ftn0)then
         do k=j+1,mcm-1
            l=l+1
            do idim=1,dim
               rpn(idim,k,ip)=rxp(idim,l)
            enddo
         enddo
      else
         do k=j+1,ftn0
            l=l+1
            do idim=1,dim
               rpn(idim,k,ip)=rxp(idim,l)
            enddo
         enddo
         do k=1,mcm-1
            l=l+1
            do idim=1,dim
               rpn(idim,k,nip)=rxp(idim,l)
            enddo
         enddo
      endif
      do idim=1,dim
         rpn(idim,ftn0+1,ip)=rpn(idim,1,nip)
         rpn(idim,0,nip)=rpn(idim,ftn0,ip)
      enddo

      return
      end


      subroutine swapn(j,ip,nip,kp,nkp)
      implicit none
!     swap the final part of the paths
      INCLUDE       'mc-sph.par'

      integer*8 ip,nip,kp,nkp,j,k,idim
      integer*8 mcm
      real*8    rr

      mcm = j+mbm-floor((j+mbm-.1)/ftn0)*ftn0

      if(j+mbm.le.ftn0)then
         do k=mcm,ftn0
            do idim=1,dim
               rr=rpn(idim,k,ip)
               rpn(idim,k,ip)=rpn(idim,k,kp)
               rpn(idim,k,kp)=rr
            enddo
         enddo
         do idim=1,dim
            rpn(idim,ftn0+1,ip)=rpn(idim,1,nkp)
            rpn(idim,ftn0+1,kp)=rpn(idim,1,nip)
            rpn(idim,0,nip)=rpn(idim,ftn0,kp)
            rpn(idim,0,nkp)=rpn(idim,ftn0,ip)
         enddo
      endif
      return
      end


      subroutine bridge(x0,x1,std,xnew,out)
      implicit none
!     sample m gaussians with std from xnew(0)=x0 to xnew(ftn0)=x1
      INCLUDE       'mc-sph.par'

      integer*8 l1,l2,l3,j,idim,out
      real*8 std,d,s,xi
      real*8 x0(mdim),x1(mdim),xnew(mdim,0:n)

      l3=mbm
      do idim=1,dim
         xnew(idim,0)=x0(idim)
         xnew(idim,l3)=x0(idim)+(x1(idim)-x0(idim))-
     :         dnint((x1(idim)-x0(idim))/sigm(idim))*sigm(idim)
      enddo
      do j=1,mbm-1
         l1=j-1
         l2=j
         s=std*(dble(l3-l2)/dble(l3-l1))**0.5d0
         do idim=1,dim
            d=xnew(idim,l3)-xnew(idim,l1)
            d=d-dnint(d/sigm(idim))*sigm(idim)
            xnew(idim,j)=xnew(idim,l1)+d/dble(l3-l1)+xi(s)
            xnew(idim,j)=xnew(idim,j)-
     :           dnint(xnew(idim,j)/sigm(idim))*sigm(idim)
         enddo
      enddo
      return
      end


      function xi(std)
!     sample a gaussian with standard deviation std (box-muller method)
      implicit none
      real*8 xi,std,pi,ranf
      data pi/3.14159265358979323846264338328d0/
      xi=cos(pi*ranf(0.d0))*std*sqrt(-2.d0*log(tiny(pi)+ranf(0.d0)))
      return
      end


      SUBROUTINE DISTR(NN,NORM)
      IMPLICIT NONE
C     WRITES ON FORT.10 THE X-POSITION DISTRIBUTION
      INCLUDE       'mc-sph.par'
```

```
      REAL*8        DS,DIST(0:1000)
      INTEGER*8     NN,NORM,I,J,K
      SAVE          DIST

      DS=SIGMA/NN

      DO I=0,NN
         DO J=1,FTNO
            DO K=1,NP
               IF(-SIGMA/2+(I-.5)*DS.LT.RX(1,J,K).AND.
     :              RX(1,J,K).LT.-SIGMA/2+(I+.5)*DS) THEN
                 DIST(I)=DIST(I)+1.DO
               ENDIF
            ENDDO
         ENDDO
         WRITE(10,*) -SIGMA/2+I*DS,DIST(I)/NORM
      ENDDO

      CLOSE(UNIT=10)

      RETURN
      END


      FUNCTION FACT ( N )
      IMPLICIT NONE
C     FACTORIAL FUNCTION
      INTEGER*8 FACT,N,P,I
      P=1
      DO I=1,N
         P=P*I
      ENDDO
      FACT=P
      END



      SUBROUTINE SUMUP (POTK, OVRLAP, KE, V, W)
      IMPLICIT NONE
C     ****************************************************************
C     ** CALCULATES THE TOTAL ENERGY                                **
C     **                                                            **
C     ** USAGE:                                                     **
C     **                                                            **
C     ** THE SUBROUTINE RETURNS THE TOTAL ENERGY AT THE             **
C     ** BEGINNING AND END OF THE RUN.                              **
C     ****************************************************************
      INCLUDE       'mc-sph.par'

      REAL*8        V, KE, VV, KK
      LOGICAL       OVRLAP
      CHARACTER     POTK*(*)

      REAL*8        RXII, RXIJ, RYIJ, RZIJ
      REAL*8        VIJ, WIJ, RIJSQ, W, WW

      INTEGER*8     TAU, I, J, IDIM

C     POTENTIAL ACTION

      VV      = 0.0
      WW      = 0.0

C     ** LOOP OVER ALL THE PAIRS IN THE LIQUID **

      DO TAU = 1, FTNO
         DO 100 I = 1, NP - 1
            DO 99 J = I + 1, NP
C               RIJSQ = 0.d0
C               DO IDIM = 1, DIM
C                  RXIJ = RX(IDIM,TAU,I) - RX(IDIM,TAU,J)
C     ** MINIMUM IMAGE THE PAIR SEPARATIONS **
C                  RXIJ = RXIJ -
C     :                 DNINT ( RXIJ/SIGM(IDIM) )*SIGM(IDIM)
C                  RIJSQ = RIJSQ + RXIJ * RXIJ
C               ENDDO

C     ** EUCLIDEAN DISTANCE
               RXIJ  = COS(RX(1,TAU,I))*COS(RX(1,TAU,J))*
     :              COS(RX(2,TAU,I))*COS(RX(2,TAU,J))
               RYIJ  = COS(RX(1,TAU,I))*COS(RX(1,TAU,J))*
     :              SIN(RX(2,TAU,I))*SIN(RX(2,TAU,J))
               RZIJ  = SIN(RX(1,TAU,I))*SIN(RX(1,TAU,J))
               RIJSQ = RXIJ + RYIJ + RZIJ
               RIJSQ = RADIUS**2*(2*(1.-RIJSQ))


               CALL POT (RIJSQ, VIJ, WIJ, POTK)
               VV   = VV + VIJ
C               WW   = WW + WIJ
 99         CONTINUE
            VV = VV - LOG(ABS(RADIUS**2*COS(RX(1,TAU,I))))/LS
            WW = WW - LOG(ABS(RADIUS**2*COS(RX(1,TAU,I))))/LS
 100     CONTINUE
         VV = VV - LOG(ABS(RADIUS**2*COS(RX(1,TAU,NP))))/LS
         WW = WW - LOG(ABS(RADIUS**2*COS(RX(1,TAU,NP))))/LS
      ENDDO
      V = VV
      W = WW


C     KINETIC ACTION

      KK     = 0.0

      DO TAU = 1, FTNO
         DO I = 1, NP
            RXII = RX(1,TAU,I)-RX(1,TAU+1,I)
            RXII = RXII - DNINT ( RXII/SIGM(1) )*SIGM(1)
            KK=KK+FTM*(RADIUS*RXII)**2./(2.DO*LS**2.)

            RXII = RX(2,TAU,I)-RX(2,TAU+1,I)
            RXII = RXII - DNINT ( RXII/SIGM(2) )*SIGM(2)
            KK=KK+FTM*(RADIUS*COS(RX(1,TAU,I))*RXII)**2./(2.DO*LS**2.)
         ENDDO
      ENDDO
      KE = KK

      RETURN
      END


      SUBROUTINE SUMUPKE ( KE )
      IMPLICIT NONE
C     ****************************************************************
C     ** CALCULATES THE TOTAL ENERGY                                **
C     **                                                            **
C     ** USAGE:                                                     **
C     **                                                            **
C     ** THE SUBROUTINE RETURNS THE TOTAL ENERGY AT THE             **
C     ** BEGINNING AND END OF THE RUN.                              **
C     ****************************************************************
      INCLUDE       'mc-sph.par'

      REAL*8        KE, KK, RXII
      INTEGER*8     TAU, I

C     KINETIC ACTION

      KK     = 0.0

      DO TAU = 1, FTNO
         DO I = 1, NP
            RXII = RX(1,TAU,I)-RX(1,TAU+1,I)
            RXII = RXII - DNINT ( RXII/SIGM(1) )*SIGM(1)
            KK=KK+FTM*(RADIUS*RXII)**2./(2.DO*LS**2.)

            RXII = RX(2,TAU,I)-RX(2,TAU+1,I)
            RXII = RXII - DNINT ( RXII/SIGM(2) )*SIGM(2)
            KK=KK+FTM*(RADIUS*COS(RX(1,TAU,I))*RXII)**2./(2.DO*LS**2.)
         ENDDO
      ENDDO
      KE = KK

      RETURN
      END


      SUBROUTINE PENERGY ( POTK, RXI, I, TAU,
     :    V, W )
      IMPLICIT NONE
C     ****************************************************************
C     ** RETURNS THE POTENTIAL ENERGY OF ATOM I WITH ALL OTHER ATOMS.  **
C     **                                                            **
C     ** PRINCIPAL VARIABLES:                                       **
C     **                                                            **
C     ** INTEGER I                THE ATOM OF INTEREST              **
C     ** INTEGER NP               THE NUMBER OF ATOMS               **
C     ** INTEGER TAU              THE TIMESTEP                      **
C     ** REAL*8    RX, RY, RZ       THE ATOM POSITIONS              **
C     ** REAL*8    RXI,RYI,RZI      THE COORDINATES OF ATOM I        **
C     ** REAL*8    V                THE POTENTIAL ENERGY OF ATOM I   **
C     ** REAL*8    W                THE VIRIAL OF ATOM I             **
C     **                                                            **
C     ** USAGE:                                                     **
C     **                                                            **
C     ** THIS SUBROUTINE IS USED TO CALCULATE THE CHANGE OF ENERGY     **
C     ** DURING A TRIAL MOVE OF ATOM I. IT IS CALLED BEFORE AND        **
C     ** AFTER THE RANDOM DISPLACEMENT OF I.                        **
C     ****************************************************************
      INCLUDE     'mc-sph.par'

      REAL*8        RXI(MDIM), V, W
      INTEGER*8     I, J, TAU, IDIM
      CHARACTER     POTK*(*)

      REAL*8        RXIJ, RYIJ, RZIJ, RIJSQ, VIJ, WIJ

      V      = 0.0
      W      = 0.0

C     ** LOOP OVER ALL MOLECULES EXCEPT I  **

      DO J = 1, NP

         IF ( I .NE. J ) THEN
            RIJSQ = 0.d0
C              DO IDIM = 1, DIM
C                 RXIJ = RXI(IDIM) - RX(IDIM,TAU,J)

C                 RXIJ = RXIJ -
C     :              DNINT ( RXIJ/SIGM(IDIM) )*SIGM(IDIM)
C                 RIJSQ = RIJSQ + RXIJ * RXIJ
C              ENDDO

            RXIJ  = COS(RXI(1))*COS(RX(1,TAU,J))*
     :           COS(RXI(2))*COS(RX(2,TAU,J))
            RYIJ  = COS(RXI(1))*COS(RX(1,TAU,J))*
     :           SIN(RXI(2))*SIN(RX(2,TAU,J))
            RZIJ  = SIN(RXI(1))*SIN(RX(1,TAU,J))
            RIJSQ = RXIJ + RYIJ + RZIJ
            RIJSQ = RADIUS**2*(2*(1.-RIJSQ))
            CALL POT (RIJSQ, VIJ, WIJ, POTK)
            V   = V + VIJ
C              W   = W + WIJ
         ENDIF

      ENDDO

      V = V - LOG(ABS(RADIUS**2*COS(RXI(1))))/LS
      W = W - LOG(ABS(RADIUS**2*COS(RXI(1))))/LS

      RETURN
```

```
      END


      SUBROUTINE PPPNERGY ( POTK, RXI, I, RXJ, J,
     :    TAU, V, W )
      IMPLICIT NONE
C     ****************************************************************
C     ** RETURNS THE POTENTIAL ENERGY OF ATOMS I AND J WITH           **
C     ** ALL OTHER ATOMS PLUS THE ONE BETWEEN I AND J                 **
C     **                                                             **
C     ** PRINCIPAL VARIABLES:                                        **
C     **                                                             **
C     ** INTEGER I, J             THE ATOMS OF INTEREST              **
C     ** INTEGER NP               THE NUMBER OF ATOMS                **
C     ** INTEGER TAU              THE TIMESTEP                       **
C     ** REAL*8    RX, RY, RZ       THE ATOM POSITIONS               **
C     ** REAL*8    RXI,RYI,RZI      THE COORDINATES OF ATOM I         **
C     ** REAL*8    RXJ,RYJ,RZJ      THE COORDINATES OF ATOM J         **
C     ** REAL*8    V                THE POTENTIAL ENERGY OF ATOM I    **
C     ** REAL*8    W                THE VIRIAL OF ATOM I              **
C     **                                                             **
C     ** USAGE:                                                      **
C     **                                                             **
C     ** THIS SUBROUTINE IS USED TO CALCULATE THE CHANGE OF ENERGY    **
C     ** DURING A TRIAL MOVE OF ATOM I. IT IS CALLED BEFORE AND       **
C     ** AFTER THE RANDOM DISPLACEMENT OF I.                         **
C     ****************************************************************
      INCLUDE     'mc-sph.par'

      REAL*8        RXI(MDIM), RXJ(MDIM), V, W
      INTEGER*8     I, J, K, TAU, IDIM
      CHARACTER     POTK*(*)

      REAL*8        RXIJ, RYIJ, RZIJ, RIJSQ, VIJ, WIJ

      V      = 0.0
      W      = 0.0

C     ** LOOP OVER ALL MOLECULES EXCEPT I AND J **

      DO 100 K = 1, NP
         IF ( K .NE. I .AND. K .NE. J ) THEN
            RIJSQ = 0.d0
C            DO IDIM = 1, DIM
C               RXIJ = RXI(IDIM) - RX(IDIM,TAU,K)

C               RXIJ = RXIJ -
C     :              DNINT ( RXIJ/SIGM(IDIM) )*SIGM(IDIM)
C               RIJSQ = RIJSQ + RXIJ * RXIJ
C            ENDDO

C     ** EUCLIDEAN DISTANCE
            RXIJ  = COS(RXI(1))*COS(RX(1,TAU,K))*
     :           COS(RXI(2))*COS(RX(2,TAU,K))
            RYIJ  = COS(RXI(1))*COS(RX(1,TAU,K))*
     :           SIN(RXI(2))*SIN(RX(2,TAU,K))
            RZIJ  = SIN(RXI(1))*SIN(RX(1,TAU,K))
            RIJSQ = RXIJ + RYIJ + RZIJ
            RIJSQ = RADIUS**2*(2*(1.-RIJSQ))

            CALL POT (RIJSQ, VIJ, WIJ, POTK)
            V   = V + VIJ
C            W   = W + WIJ

            IF ( I .NE. J ) THEN
               RIJSQ = 0.d0
C               DO IDIM = 1, DIM
C                  RXIJ = RXJ(IDIM) - RX(IDIM,TAU,K)

C                  RXIJ = RXIJ -
C     :                 DNINT ( RXIJ/SIGM(IDIM) )*SIGM(IDIM)
C                  RIJSQ = RIJSQ + RXIJ * RXIJ
C               ENDDO

C     ** EUCLIDEAN DISTANCE
               RXIJ  = COS(RXJ(1))*COS(RX(1,TAU,K))*
     :              COS(RXJ(2))*COS(RX(2,TAU,K))
               RYIJ  = COS(RXJ(1))*COS(RX(1,TAU,K))*
     :              SIN(RXJ(2))*SIN(RX(2,TAU,K))
               RZIJ  = SIN(RXJ(1))*SIN(RX(1,TAU,K))
               RIJSQ = RXIJ + RYIJ + RZIJ
               RIJSQ = RADIUS**2*(2*(1.-RIJSQ))

               CALL POT (RIJSQ, VIJ, WIJ, POTK)
               V   = V + VIJ
C               W   = W + WIJ

            ENDIF

         ENDIF

 100     CONTINUE

C        RETURN

      V = V - LOG(ABS(RADIUS**2*COS(RXI(1))))/LS
      W = W - LOG(ABS(RADIUS**2*COS(RXI(1))))/LS

      IF (I .NE. J) THEN
         RIJSQ = 0.d0
C          DO IDIM = 1, DIM
C             RXIJ = RXI(IDIM) - RXJ(IDIM)

C             RXIJ = RXIJ -
C     :            DNINT ( RXIJ/SIGM(IDIM) )*SIGM(IDIM)
C             RIJSQ = RIJSQ + RXIJ * RXIJ
C          ENDDO

C     ** EUCLIDEAN DISTANCE
          RXIJ  = COS(RXI(1))*COS(RXJ(1))*
     :         COS(RXI(2))*COS(RXJ(2))
          RYIJ  = COS(RXI(1))*COS(RXJ(1))*
     :         SIN(RXI(2))*SIN(RXJ(2))
          RZIJ  = SIN(RXI(1))*SIN(RXJ(1))
          RIJSQ = RXIJ + RYIJ + RZIJ
          RIJSQ = RADIUS**2*(2*(1.-RIJSQ))

          CALL POT (RIJSQ, VIJ, WIJ, POTK)
          V   = V + VIJ
C           W   = W + WIJ
          V = V - LOG(ABS(RADIUS**2*COS(RXJ(1))))/LS
          W = W - LOG(ABS(RADIUS**2*COS(RXJ(1))))/LS
      ENDIF

      RETURN
      END


      SUBROUTINE PPNERGY ( POTK, RPI, I, RPJ, J,
     :    TAU, V, W )
      IMPLICIT NONE
C     ****************************************************************
C     ** RETURNS THE POTENTIAL ENERGY OF ATOMS I AND J WITH           **
C     ** ALL OTHER ATOMS PLUS THE ONE BETWEEN I AND J                 **
C     **                                                             **
C     ** PRINCIPAL VARIABLES:                                        **
C     **                                                             **
C     ** INTEGER I, J             THE ATOMS OF INTEREST              **
C     ** INTEGER NP               THE NUMBER OF ATOMS                **
C     ** INTEGER TAU              THE TIMESTEP                       **
C     ** REAL*8    RX, RY, RZ       THE ATOM POSITIONS               **
C     ** REAL*8    RXI,RYI,RZI      THE COORDINATES OF ATOM I         **
C     ** REAL*8    RXJ,RYJ,RZJ      THE COORDINATES OF ATOM J         **
C     ** REAL*8    V                THE POTENTIAL ENERGY OF ATOM I    **
C     ** REAL*8    W                THE VIRIAL OF ATOM I              **
C     **                                                             **
C     ** USAGE:                                                      **
C     **                                                             **
C     ** THIS SUBROUTINE IS USED TO CALCULATE THE CHANGE OF ENERGY    **
C     ** DURING A TRIAL MOVE OF ATOM I. IT IS CALLED BEFORE AND       **
C     ** AFTER THE RANDOM DISPLACEMENT OF I.                         **
C     ****************************************************************
      INCLUDE     'mc-sph.par'

      REAL*8        RPI(MDIM),RPJ(MDIM),RXI(MDIM),RXJ(MDIM), V, W
      INTEGER*8     I, J, K, TAU, IDIM
      CHARACTER     POTK*(*)

      REAL*8        RXIJ, RYIJ, RZIJ, RIJSQ, VIJ, WIJ

C     MAPPING
      RXI(1)=RPI(1)
      RXI(2)=RPI(2)
      RXJ(1)=RPJ(1)
      RXJ(2)=RPJ(2)

      V      = 0.0
      W      = 0.0

C     ** LOOP OVER ALL MOLECULES EXCEPT I AND J **

      DO 100 K = 1, NP
         IF ( K .NE. I .AND. K .NE. J ) THEN
            RIJSQ = 0.d0
C             DO IDIM = 1, DIM
C                RXIJ = RXI(IDIM) - RX(IDIM,TAU,K)

C                RXIJ = RXIJ -
C     :               DNINT ( RXIJ/SIGM(IDIM) )*SIGM(IDIM)
C             RIJSQ = RIJSQ + RXIJ * RXIJ
C             ENDDO

C     ** EUCLIDEAN DISTANCE
            RXIJ  = COS(RXI(1))*COS(RX(1,TAU,K))*
     :           COS(RXI(2))*COS(RX(2,TAU,K))
            RYIJ  = COS(RXI(1))*COS(RX(1,TAU,K))*
     :           SIN(RXI(2))*SIN(RX(2,TAU,K))
            RZIJ  = SIN(RXI(1))*SIN(RX(1,TAU,K))
            RIJSQ = RXIJ + RYIJ + RZIJ
            RIJSQ = RADIUS**2*(2*(1.-RIJSQ))

            CALL POT (RIJSQ, VIJ, WIJ, POTK)
            V   = V + VIJ
C             W   = W + WIJ

            IF ( I .NE. J ) THEN
               RIJSQ = 0.d0
C                DO IDIM = 1, DIM
C                   RXIJ = RXJ(IDIM) - RX(IDIM,TAU,K)

C                   RXIJ = RXIJ -
C     :                  DNINT ( RXIJ/SIGM(IDIM) )*SIGM(IDIM)
C                   RIJSQ = RIJSQ + RXIJ * RXIJ
C                ENDDO

C     ** EUCLIDEAN DISTANCE
               RXIJ  = COS(RXJ(1))*COS(RX(1,TAU,K))*
     :              COS(RXJ(2))*COS(RX(2,TAU,K))
               RYIJ  = COS(RXJ(1))*COS(RX(1,TAU,K))*
     :              SIN(RXJ(2))*SIN(RX(2,TAU,K))
               RZIJ  = SIN(RXJ(1))*SIN(RX(1,TAU,K))
               RIJSQ = RXIJ + RYIJ + RZIJ
               RIJSQ = RADIUS**2*(2*(1.-RIJSQ))

               CALL POT (RIJSQ, VIJ, WIJ, POTK)
               V   = V + VIJ
C                W   = W + WIJ

            ENDIF

         ENDIF
```

```
 100    CONTINUE

C        RETURN

         V = V - LOG(ABS(RADIUS**2*COS(RXI(1))))/LS
         W = W - LOG(ABS(RADIUS**2*COS(RXI(1))))/LS

         IF (I .NE. J) THEN
            RIJSQ = 0.d0
C            DO IDIM = 1, DIM
C               RXIJ = RXI(IDIM) - RXJ(IDIM)

C               RXIJ = RXIJ -
C     :              DNINT ( RXIJ/SIGM(IDIM) )*SIGM(IDIM)
C               RIJSQ = RIJSQ + RXIJ * RXIJ
C            ENDDO

C     ** EUCLIDEAN DISTANCE
            RXIJ  = COS(RXI(1))*COS(RXJ(1))*
     :           COS(RXI(2))*COS(RXJ(2))
            RYIJ  = COS(RXI(1))*COS(RXJ(1))*
     :           SIN(RXI(2))*SIN(RXJ(2))
            RZIJ  = SIN(RXI(1))*SIN(RXJ(1))
            RIJSQ = RXIJ + RYIJ + RZIJ
            RIJSQ = RADIUS**2*(2*(1.-RIJSQ))

            CALL POT (RIJSQ, VIJ, WIJ, POTK)
            V   = V + VIJ
C            W   = W + WIJ
            V = V - LOG(ABS(RADIUS**2*COS(RXJ(1))))/LS
            W = W - LOG(ABS(RADIUS**2*COS(RXJ(1))))/LS
         ENDIF

         RETURN
         END


         SUBROUTINE KENERGY ( RXI, I, TAU, KE )
         IMPLICIT NONE
C    ****************************************************************
C    ** RETURNS THE KINETIC ENERGY OF ATOM I WITH ALL OTHER ATOMS.  **
C    **                                                            **
C    ** PRINCIPAL VARIABLES:                                       **
C    **                                                            **
C    ** INTEGER I                THE ATOM OF INTEREST              **
C    ** INTEGER NP               THE NUMBER OF ATOMS               **
C    ** INTEGER TAU              THE TIMESTEP                      **
C    ** REAL*8    RX, RY, RZ       THE ATOM POSITIONS              **
C    ** REAL*8    RXI,RYI,RZI      THE COORDINATES OF ATOM I       **
C    ** REAL*8    KE               THE KINETIC ENERGY OF ATOM I    **
C    **                                                            **
C    ** USAGE:                                                     **
C    **                                                            **
C    ** THIS SUBROUTINE IS USED TO CALCULATE THE CHANGE OF ENERGY  **
C    ** DURING A TRIAL MOVE OF ATOM I. IT IS CALLED BEFORE AND     **
C    ** AFTER THE RANDOM DISPLACEMENT OF I.                        **
C    ****************************************************************
         INCLUDE     'mc-sph.par'

         REAL*8        RXI(MDIM), KE
         REAL*8        RXIP1, RXIS1, RXIP2, RXIS2, RXII
         INTEGER*8     I, TAU, IDIM

         KE = 0.d0

         RXIP1 = RX(1,TAU-1,I)
         RXIS1 = RX(1,TAU+1,I)
         RXII = RXI(1) - RXIP1
         RXII = RXII - DNINT ( RXII/SIGM(1) )*SIGM(1)
         KE=KE+0.5*FTM*(RADIUS*RXII/LS)**2.
         RXII = RXI(1) - RXIS1
         RXII = RXII - DNINT ( RXII/SIGM(1) )*SIGM(1)
         KE=KE+0.5*FTM*(RADIUS*RXII/LS)**2.

         RXIP2 = RX(2,TAU-1,I)
         RXIS2 = RX(2,TAU+1,I)
         RXII = RXI(2) - RXIP2
         RXII = RXII - DNINT ( RXII/SIGM(2) )*SIGM(2)
         KE=KE+0.5*FTM*(RADIUS*COS(RXIP1)*RXII/LS)**2.
         RXII = RXI(2) - RXIS2
         RXII = RXII - DNINT ( RXII/SIGM(2) )*SIGM(2)
         KE=KE+0.5*FTM*(RADIUS*COS(RXI(1))*RXII/LS)**2.

         RETURN
         END


         SUBROUTINE KKNERGY ( RXI, IP, TAU, KE )
         IMPLICIT NONE
C    ****************************************************************
C    ** RETURNS THE KINETIC ENERGY OF ATOM I WITH ALL OTHER ATOMS.  **
C    **                                                            **
C    ** PRINCIPAL VARIABLES:                                       **
C    **                                                            **
C    ** INTEGER I                THE ATOM OF INTEREST              **
C    ** INTEGER NP               THE NUMBER OF ATOMS               **
C    ** INTEGER TAU              THE TIMESTEP                      **
C    ** REAL*8    RX, RY, RZ       THE ATOM POSITIONS              **
C    ** REAL*8    RXI,RYI,RZI      THE COORDINATES OF ATOM I       **
C    ** REAL*8    KE               THE KINETIC ENERGY OF ATOM I    **
C    **                                                            **
C    ** USAGE:                                                     **
C    **                                                            **
C    ** THIS SUBROUTINE IS USED TO CALCULATE THE CHANGE OF ENERGY  **
C    ** DURING A TRIAL MOVE OF ATOM I. IT IS CALLED BEFORE AND     **
C    ** AFTER THE RANDOM DISPLACEMENT OF I.                        **
C    ****************************************************************
         INCLUDE     'mc-sph.par'

         REAL*8        RXI(MDIM), KE
         REAL*8        RXIS, RXII
         INTEGER*8     IP, TAU, IDIM

         KE = 0.d0

         RXIS = RX(1,TAU+1,IP)
         RXII = RXI(1) - RXIS
         RXII = RXII - DNINT ( RXII/SIGM(1) )*SIGM(1)
         KE=KE+0.5*FTM*(RADIUS*RXII/LS)**2.

         RXIS = RX(2,TAU+1,IP)
         RXII = RXI(2) - RXIS
         RXII = RXII - DNINT ( RXII/SIGM(2) )*SIGM(2)
         KE=KE+0.5*FTM*(RADIUS*COS(RXI(1))*RXII/LS)**2.

         RETURN
         END


         SUBROUTINE CWIND ( WIND )
         IMPLICIT NONE
C    ****************************************************************
C    ** RETURNS THE WINDING NUMBER.                                **
C    ****************************************************************
         INCLUDE     'mc-sph.par'

         REAL*8        WIND, WI, SURF
         REAL*8        RX11, RX21, RX31, RX12, RX22, RX32
         INTEGER*8     I, J, IDIM

         WIND = 0.d0
         SURF = 4*PI*RADIUS**2
         DO I = 1, NP
            DO J = 1, FTNO - 1
               RX11 = RX(1,J  ,I)
               RX21 = RX(1,J+1,I)
               RX31 = RX(1,J+2,I)
               RX12 = RX(2,J  ,I)
               RX22 = RX(2,J+1,I)
               RX32 = RX(2,J+2,I)
               WI = ((RX21-RX11)*(RX32-RX22)-(RX22-RX12)*(RX31-RX21))*
     :              RADIUS**2*COS(RX21)
               WIND = WIND + WI
            ENDDO
         ENDDO
         IF (WIND.GT.SURF) WIND = SURF
         WIND = WIND**2

 20      CONTINUE
         RETURN
         END


         REAL*8 FUNCTION RANF ( DUMMY )
         IMPLICIT NONE
C    ****************************************************************
C    ** RETURNS A UNIFORM RANDOM VARIATE IN THE RANGE 0 TO 1.       **
C    **                                                            **
C    **                ***************                             **
C    **                **  WARNING  **                             **
C    **                ***************                             **
C    **                                                            **
C    ** GOOD RANDOM NUMBER GENERATORS ARE MACHINE SPECIFIC.        **
C    ** PLEASE USE THE ONE RECOMMENDED FOR YOUR MACHINE.           **
C    **                                                            **
C    ** RAND(FLAG) returns a pseudo-random number from a uniform   **
C    ** distribution between 0 and 1. If FLAG is 0, the next number **
C    ** in the current sequence is returned; if FLAG is 1, the     **
C    ** generator is restarted by CALL SRAND(0); if FLAG has any   **
C    ** other value, it is used as a new seed with SRAND.          **
C    ****************************************************************
         INTEGER*8   L, C, M
         PARAMETER ( L = 1029, C = 221591, M = 1048576 )
         INTEGER*8   SEED
         SAVE        SEED
         DATA        SEED / 0 /
         REAL*8      DUMMY

C        SEED = MOD ( SEED * L + C, M )
C        RANF = DBLE( SEED ) / M

         RANF = RAND ( )

         RETURN
         END


         SUBROUTINE READCN ( CNFILE )
         IMPLICIT NONE
C    ****************************************************************
C    ** SUBROUTINE TO READ IN THE CONFIGURATION FROM UNIT 10        **
C    ****************************************************************
         INCLUDE     'mc-sph.par'

         CHARACTER   CNFILE*(*), HASH*1, MYFMT*77

         INTEGER*8   CNUNIT, I, J, NNP, NI, IDIM
         PARAMETER ( CNUNIT = 10 )

C    ****************************************************************

         OPEN ( UNIT = CNUNIT, FILE = CNFILE, STATUS = 'OLD' )
         WRITE(MYFMT,'(A,I10,A)') '(I3,3X,',DIM,'(F12.6,3X))'

         READ ( CNUNIT,* ) HASH, FTNO, NNP
         IF ( NNP .NE. NP ) STOP 'N ERROR IN READCN'

         DO 100 I = 1, NNP
            READ ( CNUNIT,* ) HASH, NEXT(I)
            NI = NEXT(I)
            DO 90 J = 1, FTNO+1
               READ ( CNUNIT, MYFMT) LEXT(I), (RX(IDIM,J,I),IDIM=1,DIM)
```

```
 90        ENDDO
           READ ( CNUNIT,13 )
           READ ( CNUNIT,13 )
           DO IDIM = 1, DIM
              RX(IDIM,0,NI) = RX(IDIM,FTN0,I)
           ENDDO
 100    ENDDO
 13     FORMAT(A2)

        CLOSE ( UNIT = CNUNIT )

        RETURN
        END


        SUBROUTINE WRITCN ( CNFILE )
        IMPLICIT NONE
C    *****************************************************************
C    ** SUBROUTINE TO WRITE OUT THE CONFIGURATION TO UNIT 10          **
C    *****************************************************************
        INCLUDE     'mc-sph.par'

        CHARACTER   CNFILE*(*), MYFMT*77

        INTEGER*8   CNUNIT, I, J, IDIM
        REAL*8      R
        PARAMETER ( CNUNIT = 10 )

C   *****************************************************************

        OPEN ( UNIT = CNUNIT, FILE = CNFILE, STATUS = 'UNKNOWN' )
        WRITE(MYFMT,'(A,I10,A)') '(I3,3X,',DIM,'(F12.6,3X))'

        WRITE(*,*) 'output to file ------------------------------------
     :--------------'
        CALL FLUSH ( CNUNIT )
        WRITE ( CNUNIT,* ) '#',FTN0,NP

        DO I = 1, NP
           WRITE ( CNUNIT,* ) '#', NEXT(I)
           DO J = 1, FTN0+1
C             WRITE ( CNUNIT, MYFMT ) LEXT(I),
C     :    (RX(IDIM,J,I)-DNINT(RX(IDIM,J,I)/SIGMA)*SIGMA,IDIM=1,DIM)
              WRITE ( CNUNIT, MYFMT ) LEXT(I),
     :             (RX(IDIM,J,I),IDIM=1,DIM)
           ENDDO
           WRITE ( CNUNIT,13 ) ''
           WRITE ( CNUNIT,13 ) ''
        ENDDO
 13     FORMAT(A2)

        CLOSE ( UNIT = CNUNIT )

        RETURN
        END


        SUBROUTINE INITCNSPH ( CNFILE )
        IMPLICIT NONE
C    *****************************************************************
C    ** SUBROUTINE TO INITIALIZE THE CONFIGURATION TO UNIT 10         **
C    *****************************************************************
        INCLUDE     'mc-sph.par'

        REAL*8      RANF
        CHARACTER   CNFILE*(*), MYFMT*77

        INTEGER*8   CNUNIT, I, J, IDIM
        REAL*8      RXI(DIM)
        PARAMETER ( CNUNIT = 10 )

C   *****************************************************************

        OPEN ( UNIT = CNUNIT, FILE = CNFILE, STATUS = 'UNKNOWN' )
        WRITE(MYFMT,'(A,I10,A)') '(I3,3X,',DIM,'(F12.6,3X))'

        WRITE ( CNUNIT,* ) '#',FTN0,NP

        DO I = 1, NP
           WRITE ( CNUNIT,* ) '#', NEXT(I)
           RXI(1) = 0.
           RXI(2) = SIGM(2)*(RANF(0.d0)-.5)
           DO J = 1, FTN0+1
              WRITE ( CNUNIT, MYFMT ) I, RXI(1), RXI(2)
           ENDDO
           WRITE ( CNUNIT,13 ) ''
           WRITE ( CNUNIT,13 ) ''
        ENDDO
 13     FORMAT(A2)

        CLOSE ( UNIT = CNUNIT )

        RETURN
        END


        SUBROUTINE POT (RIJSQ, VIJ, WIJ, POTK)
        IMPLICIT NONE
C    *****************************************************************
C    ** SUBROUTINE FOR THE PAIR POTENTIAL                            **
C    *****************************************************************
        INCLUDE     'mc-sph.par'

        CHARACTER POTK*(*)
        REAL*8    RIJ, RIJSQ, RMIN, RMSQ, RCSQ, SR2, SR6, VIJ, WIJ
        REAL*8    COU3, QHS3, F
        PARAMETER ( COU3 = 4.760154727959107013287634700 57d0 )
        PARAMETER ( QHS3 = 24.d0 )

C       PI = ACOS(-1.d0)

        VIJ = 0.d0
        WIJ = 0.d0
        IF (DIM .GT. 3) THEN
           WRITE(*,*) "DIM > 3 !!!"
           STOP
        ENDIF
        IF (POTK .EQ. 'FREE') RETURN

        IF (POTK .EQ. 'LJ') THEN
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     W. L. McMillan, Phys. Rev. A 138, 442 (1965)     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C          RMIN = 2.d0**(1./6)
C          RMIN = EPSA
C          RMIN = TINY(PI)
           RMIN = 0.d0
           RIJSQ = RIJSQ / (SIG * SIG)
           RMSQ = RMIN*RMIN
           RCSQ = RCUT*RCUT
           IF (RIJSQ .LT. RMSQ) THEN
              RIJSQ = RMSQ
              SR2 = 1.d0 / RIJSQ
              SR6 = SR2 * SR2 * SR2
              VIJ = SR6 * ( SR6 - 1.d0 )
              WIJ = SR6 * ( SR6 - 5.d-1 )
              VIJ = 4.d0 * EPS * VIJ
              WIJ = 48.d0 * EPS * WIJ / 3.d0
           ELSEIF (RIJSQ .GT. RMSQ .AND. RIJSQ .LT. RCSQ) THEN
              SR2 = 1.d0 / RIJSQ
              SR6 = SR2 * SR2 * SR2
              VIJ = SR6 * ( SR6 - 1.d0 )
              WIJ = SR6 * ( SR6 - 5.d-1 )
              VIJ = 4.d0 * EPS * VIJ
              WIJ = 48.d0 * EPS * WIJ / 3.d0
           ENDIF
        ELSEIF (POTK .EQ. 'BUMP') THEN
           RIJ = SQRT( RIJSQ )
           IF (RIJ .LE. SIG) THEN
              VIJ = EPSR
           ENDIF
        ELSEIF (POTK .EQ. 'PSW') THEN
           RIJ = SQRT( RIJSQ )
           IF (RIJ .LE. SIG) THEN
              VIJ = EPSR
           ELSEIF (RIJ .LE. SIG + RCUT .AND. RIJ .GT. SIG) THEN
              VIJ = -EPSA
           ENDIF
        ELSEIF (POTK .EQ. 'COULOMB') THEN
           F = NP/(NP-1)
           RIJ = SQRT( RIJSQ )
           IF (RIJ .GT. TINY(PI)) THEN
              IF (DIM .EQ. 3) THEN
                 VIJ = -F*COU3*SIG/SIGMA/2.
                 VIJ = VIJ + SIG/RIJ
              ELSEIF (DIM .EQ. 2) THEN
                 VIJ = -F*(6.-PI+LOG(4.)-4.*LOG(SIGMA/SIG))/4.
                 VIJ = VIJ - LOG(RIJ/SIG)
              ELSEIF (DIM .EQ. 1) THEN
                 VIJ = F*SIGMA/SIG/4.
                 VIJ = VIJ - RIJ/SIG
              ENDIF
           ELSE
              RIJ = TINY(PI)
              IF (DIM .EQ. 3) THEN
                 VIJ = -F*COU3*SIG/SIGMA/2.
                 VIJ = VIJ + SIG/RIJ
              ELSEIF (DIM .EQ. 2) THEN
                 VIJ = -F*(6.-PI+LOG(4.)-4.*LOG(SIGMA/SIG))/4.
                 VIJ = VIJ - LOG(RIJ/SIG)
              ELSEIF (DIM .EQ. 1) THEN
                 VIJ = F*SIGMA/SIG/4.
                 VIJ = VIJ - RIJ/SIG
              ENDIF
           ENDIF
        ELSEIF (POTK .EQ. 'QHS') THEN
           F = NP/(NP-1)
           IF (DIM .EQ. 3) THEN
              IF (RIJSQ .LE. SIG*SIG) THEN
                 VIJ = 10.**10.
              ELSE
              VIJ = -F*QHS3*(SIGMA-SIG)/SIGMA**3/2./FTM
              VIJ = VIJ + (2.*RIJSQ+SIG**2.)/(RIJSQ-SIG*SIG)**2./2./FTM
              WIJ = -2.*RIJSQ*(RIJSQ+2.*SIG*SIG)/(RIJSQ-SIG*SIG)**3./FTM
              ENDIF
           ELSE
              WRITE(*,*) 'QHS in dimension different from 3'
              STOP
           ENDIF
        ELSEIF (POTK .EQ. 'HARMONIC') THEN
           VIJ = 0.5*RIJSQ/SIG**2.
        ENDIF

        RETURN
        END


        SUBROUTINE GOFR ( TAU, MAXBIN, HIST )
        IMPLICIT NONE
C    *****************************************************************
C    ** CALCULATES THE RADIAL DISTRIBUTION FUNCTION HISTOGRAM.       **
C    *****************************************************************
        INCLUDE     'mc-sph.par'

        REAL*8        RXIJ, RYIJ, RZIJ, RIJSQ, RIJ
        INTEGER*8     TAU, IDIM, I, J, BIN, MAXBIN, HIST(MAXBIN)

        DO I = 1, NP - 1
           DO J = I + 1, NP
C     CALCULATE HISTOGRAM FOR g(r)

C     Euclidean Distance
```

```
            RXIJ  = COS(RX(1,TAU,I))*COS(RX(1,TAU,J))*
     :           COS(RX(2,TAU,I))*COS(RX(2,TAU,J))
            RYIJ  = COS(RX(1,TAU,I))*COS(RX(1,TAU,J))*
     :           SIN(RX(2,TAU,I))*SIN(RX(2,TAU,J))
            RZIJ  = SIN(RX(1,TAU,I))*SIN(RX(1,TAU,J))
            RIJSQ = RXIJ + RYIJ + RZIJ
c            RIJ = 2*RADIUS*SIN(ACOS(RIJSQ)/2.)
            RIJ = RADIUS*SQRT(2.-2*RIJSQ)

C     Geodesic Distance
c            RIJ = RADIUS*ACOS(SIN(RX(1,TAU,I))*SIN(RX(1,TAU,J))+
c     : COS(RX(1,TAU,I))*COS(RX(1,TAU,J))*COS(RX(2,TAU,I)-RX(2,TAU,J)))

            BIN = INT ( RIJ/DELR ) + 1

            IF (BIN .LE. MAXBIN) THEN
               HIST(BIN) = HIST(BIN) + 2
            ENDIF

         ENDDO
      ENDDO

      RETURN
      END


      SUBROUTINE MAPPINGD()
      IMPLICIT NONE
C     ****************************************************************
C     ** CHANGE COORDINATES (DIRECT)                                **
C     ****************************************************************
      INCLUDE     'mc-sph.par'

      INTEGER*8 I,J

      DO I=1,NP
         DO J=0,FTNO+1
            RP(1,J,I)=RX(1,J,I)
            RP(2,J,I)=RX(2,J,I)
         ENDDO
      ENDDO

      RETURN
      END


      SUBROUTINE MAPPINGI()
      IMPLICIT NONE
C     ****************************************************************
C     ** CHANGE COORDINATES (INVERSE)                               **
C     ****************************************************************
      INCLUDE     'mc-sph.par'

      INTEGER*8 I,J

      DO I=1,NP
         DO J=0,FTNO+1
            RX(1,J,I)=RP(1,J,I)
            RX(2,J,I)=RP(2,J,I)
         ENDDO
      ENDDO

      RETURN
      END


      SUBROUTINE GAUSS(CP,GA)
      IMPLICIT NONE
C     ****************************************************************
C     ** GAUSSIAN TRANSITION PROBABILITY FOR BROWNIAN BRIDGE         **
C     ****************************************************************
      INCLUDE     'mc-sph.par'

      INTEGER*8 I,J,K,CP,MCM
      REAL*8 GA,VAR,DD

      MCM = CP+MBM-FLOOR((CP+MBM-.1)/FTNO)*FTNO
      VAR = 2*LS/FTM

      GA = 0.D0
      DO I=1,DIM
         DO J=1,NP
            IF(CP+MBM.LE.FTNO)THEN
               DO K=CP+1,MCM
                  DD=RPN(I,K,J)-RPN(I,K-1,J)
                  DD=DD-DNINT ( DD/SIGM(I) )*SIGM(I)
                  GA=GA+DD**2
                  DD=RP(I,K,J)-RP(I,K-1,J)
                  DD=DD-DNINT ( DD/SIGM(I) )*SIGM(I)
                  GA=GA-DD**2
               ENDDO
            ELSE
               DO K=CP+1,FTNO
                  DD=RPN(I,K,J)-RPN(I,K-1,J)
                  DD=DD-DNINT ( DD/SIGM(I) )*SIGM(I)
                  GA=GA+DD**2
                  DD=RP(I,K,J)-RP(I,K-1,J)
                  DD=DD-DNINT ( DD/SIGM(I) )*SIGM(I)
                  GA=GA-DD**2
               ENDDO
               DO K=1,MCM
                  DD=RPN(I,K,J)-RPN(I,K-1,J)
                  DD=DD-DNINT ( DD/SIGM(I) )*SIGM(I)
                  GA=GA+DD**2
                  DD=RP(I,K,J)-RP(I,K-1,J)
                  DD=DD-DNINT ( DD/SIGM(I) )*SIGM(I)
                  GA=GA-DD**2
               ENDDO
            ENDIF
         ENDDO
      ENDDO

      GA = EXP(GA/VAR)

      RETURN
      END


      SUBROUTINE EQUIV(A,B)
      IMPLICIT NONE
C     ****************************************************************
C     ** A = B WITH A, B TWO PATHS                                  **
C     ****************************************************************
      INCLUDE     'mc-sph.par'

      INTEGER*8 I,J,K
      REAL*8    A(MDIM,0:N,MNP),B(MDIM,0:N,MNP)

      DO I=1,DIM
         DO J=0,FTNO+1
            DO K=1,NP
               A(I,J,K)=B(I,J,K)
            ENDDO
         ENDDO
      ENDDO

      RETURN
      END
```

```
*******************************************************************************
*** mc-sph.par ****************************************************************
*******************************************************************************

     REAL*8        PI
     REAL*8        DELR
        INTEGER*8     MDIM, N, MNP

        PARAMETER ( PI = 3.14159265358979323846264338328d0 )

     PARAMETER ( MDIM = 10   )   ! maximum number of dimensions
     PARAMETER ( MNP  = 1000 )   ! maximum number of particles
     PARAMETER ( N    = 3000 )   ! maximum number of time slices
     PARAMETER ( DELR = 0.01d0 ) ! grid spacing for g(r)

     INTEGER*8     DIM
        REAL*8        RX(MDIM,0:N,MNP)
     REAL*8        RPN(MDIM,0:N,MNP),RP(MDIM,0:N,MNP)
     REAL*8        FTM, LS, SIGMA, SIGM(MDIM), RADIUS
        REAL*8        SIG, EPSA, EPSR, EPS, RCUT
        INTEGER*8     FTNO, NP
     INTEGER*8     NEXT(MNP), LEXT(MNP), MEXT(N,MNP), MBM

        ! path
        COMMON / BLOCK1 / RX, RP, RPN, DIM
        ! pair-potential parameters
        COMMON / BLOCK2 / SIG, EPSR, EPSA, EPS, RCUT
        ! mass, timestep, box edge, # timeslices, # particles, radius
        COMMON / BLOCK3 / FTM, LS, SIGMA, SIGM, FTNO, NP, RADIUS
        ! permutations
        COMMON / BLOCK4 / NEXT, LEXT, MEXT, MBM
```

```
*******************************************************************************
*** data-gr-rs.in *************************************************************
*******************************************************************************

0 number of spatial dimensions          (2 sphere)
2
1 seed of the random sequence                RAND
3
2 if bose                                   (T/F)
T
3 number of particles                    (<= MNP)
10
4 the potential                     SUBROUTINE POT
FREE
5 # time slices = 1/temperature/timestep     (< N)
50
6 mass                              (hbar = kb = 1)
1.d0
7 # of cycles                              (nstep)
500000000000000000
8 # of steps between output lines         (iprint)
100
9 # of steps between configuration saves   (isave)
100
10 # of steps for equilibration            (iequi)
100
11 # of steps for acceptance ratios       (iratio)
100
12 # of steps for rdf calculation    (icalcg < #5)
10
13 configuration file name
conf.xyz
14 0 init zero, 1 init, 2 rest conf, 3 rest full
1
15 radius of the sphere
5.d0
16 temperature                      THERMODYNAMICS
.03d0
17 maximum displacement/box edge
.05d0
18 maximum # of bridge timeslices     (> 1; <= #5)
10
19 potential cutoff distance (LJ)   SUBROUTINE POT
2.d0
20 sig    (LJ 2.566)                SUBROUTINE POT
1.d0
21 epsr   (LJ INIT)                 SUBROUTINE POT
1.d10
22 epsa                             SUBROUTINE POT
1.d0
23 eps    (LJ 10.22)                SUBROUTINE POT
```

```
1.d1
24 if only displace                    (T/F)
F
25 if only bridge                      (T/F)
F
```

AUTHOR DECLARATIONS

Conflicts of interest

None declared.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

Funding

None declared.

[1] R. Fantoni, Coherent State Path Integral Monte Carlo, Eur. Phys. J. D (2025), under review.
[2] D. M. Ceperley, Path integrals in the theory of condensed Helium, Rev. Mod. Phys. **67**, 279 (1995).
[3] D. M. Ceperley, Fermion Nodes, J. Stat. Phys. **63**, 1237 (1991).
[4] D. M. Ceperley, Path integral Monte Carlo methods for fermions, in *Monte Carlo and Molecular Dynamics of Condensed Matter Systems*, edited by K. Binder and G. Ciccotti (Editrice Compositori, Bologna, Italy, 1996).