Supplementary material for the work "Electron-Ion Path Integral Monte Carlo"

Riccardo Fantoni*
Università di Trieste, Dipartimento di Fisica, strada Costiera 11, 34151 Grignano (Trieste), Italy
(Dated: May 2, 2026)

Supplementary material to the article "Electron-Ion Path Integral Monte Carlo".

Keywords: Two-Component-Plasma; Electron-Ion Plasma; Fermi-Dirac Statistics; Restricted Path Integral Monte Carlo; Thermodynamics; Structure

I. INTRODUCTION

We here report the FORTRAN code listing of the code used in the simulations developed in the publication "Electron-Ion Path Integral Monte Carlo". This is shown in Appendix A. The model studied in that publication was that of a 1:1 electron-proton plasma. A particular Two Component Plasma (TCP) where the two species are both fermions, one about 2000 times more heavy than the other. A particular neutral binary mixture with Coulomb pair interactions.

The Restricted Path Integral Monte Carlo (RPIMC) method used in our computer experiments is the one described in Ref. [1, 2]. But the permutation sampling necessary in calculating the determinant required by the study of the Fermi-Dirac statistics was carried out proposing swaps of a randomly chosen pair of particles through the Lévy construction of a pair of Brownian bridges (see Ref. [3] or Section V.G of Ref. [1] and references therein) connecting the two paths.

In the simulation we measure some thermodynamic properties like the total kinetic and potential energy and the superfluid fraction of each species. And some structural properties like the partial Radial Distribution Functions (RDF) for the two species.

The plane waves [1] Path Integral Monte Carlo (PIMC) method used in our computer experiments is the one described in Ref. [1]. A single MC step is made of a swap of a randomly chosen pair of particles through the *Lévy construction* of two Brownian bridge between two randomly chosen timeslices on the two different paths (see Section V.G of Ref. [1] 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). This allows the permutation sampling necessary in calculating the symmetric density matrix required by the study of the Bose-Einstein statistics. We then introduced a path restriction in order to overcome the "fermions sign problem" [5] based on the sign of the free fermions density matrix calculated in subroutine `ifdm`. This results in an approximate RPIMC calculation [2] which calculates the antisymmetric density matrix required by the study of the Fermi-Dirac statistics. Of course we only apply the swap to like particles of the same species.

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 `FTNO` timeslices of length `LS=BETA/FTNO`. One usually wants to compare simulations at fixed `LS`. The multislice move requires the reconstruction of a randomly chosen number `MBM`$\leq$`MBMM`$\leq$`FNTO` of timeslices between a randomly chosen initial timeslice `CP` on a randomly chosen particle `IP` and the final 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 `FTNO` timeslices of the path of the particle `IP` chosen at random. So a single MC step requires a maximum of `2MBMM+FTNO` 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 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 these variables can be chosen in the `data-mix.in` simulation configuration input file read at startup.

The code works in any space dimension `DIM`$\leq$`MDIM` with a mixture with a number of components `NMIX`$\leq$`MMIX`. The number of particles of species `I` is `NPM(I)` and they have mass `FTM(I)`. One has to specify also the thermodynamic configuration giving the density `DENS` 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 AQHS effective potential (`QHS`) and the canonical HS (`BUMP`). Other test cases included are the harmonic potenial (`HARMONIC`), the Coulomb potential (`COULOMB`), the penetrable square well potential (`PSW`), and the Lennard-Jones potential (`LJ`). The ideal gas is obtained choosing the string `FREE` in the `data-mix.in` simulation configuration input file. The potential can be further specified through the variables `RCUT`, `SIG`, `EPSR`, `EPSA`, and `EPS` in the simulation configuration input file. For the sake of the present work we only use the Coulomb potential where `SIG(I,J)` is given by Eq. (2.8) in the

* riccardo.fantoni@scuola.istruzione.it
[1] For a coherent states PIMC see Ref. [4].

main text of the article and `EPSA(I,J)` is given by Eq. (2.9) in the main text of the article.

The simulation calculates both thermodynamic quantities and static structural properties. For the thermodynamic quantities we measure: the kinetic energy `KE`, the potential energy `V`, and the superfluid fraction of each species `SF(I)`. 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. In order to determine efficiently the superfluid fraction at low temperature, which requires large `FTN0`, it is convenient to deactivate the displace move choosing `TRUE` for `IFBRIDGE`. For the structural properties we measure the partial radial distribution function $g_{ij}(r)$ which are written in the output files `rdf11.dat`, `rdf12.dat`, `rdf22.dat`. In the current implementation of the code, the $g_{ij}(r)$ are only calculated during the displacement move every `ICALCG` timeslices.

The logical variable `IFB` is set in the simulation configuration input file. If it is set to `TRUE` permutations of particles are sampled, otherwise only Boltzmann statistics for distinguishable particles is used.

If the variable `INIT` is set to 0 it starts from a path with the particles randomly distributed uniformly with no overlaps with all timeslices copied above the first one for each particle, if it is set to 1 it reads the paths configuration from the file `CNFILE` stored on disk, if it is set to 2 it does a complete restart from the file `CNFILE` and the frozen configuration `fort.20` both stored on disk every `ISAVE` MC steps.

The variable `SEED` is also set in the simulation configuration input file `data-mix.in`. It is the positive integer seed of the pseudo random number generator.

ACKNOWLEDGMENTS

I would like to thank prof. Saverio Moroni for his support in the development of the Brownian bridge which allows the particles permutation sampling.

Appendix A: The code

This is the code used for the PIMC computer experiment. We list here the main FORTRAN code `rpimc-mix.f` with its included `mc-mix.par` parameters file. And the input data file `data-mix.in` (for cases G and H of the Table in the main text of the article) that is read at the beginning of the run.

```
********************************************************************************
***  rpimc-mix.f  ************************************************************
********************************************************************************

       PROGRAM RPIMC
       IMPLICIT NONE
C     ****************************************************************
C     ** MONTE CARLO SIMULATION PROGRAM FOR [PW] RESTRICTED PATH      **
C     ** INTEGRAL FOR A BINARY MIXTURE                                **
C     **                                                              **
C     ** space dimensions:    DIM=1,2,3                               **
C     ** number of particles: NP                                      **
C     ** number of timesteps: FTN0                                    **
C     ** units:               hbar=k_B=1                              **
C     ** BETA=1/TEMP=FTN0*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 in                             **
C     ** 'rdf11.dat', 'rdf12.dat', 'rdf22.dat'                        **
C     ****************************************************************
       INCLUDE       'mc-mix-tab.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     RS

       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(MMIX), PS, KKK, VVV, WWW
       REAL*8        RATIO, RATIOS, RATIOB
       REAL*8        WIND(MMIX,MDIM), ACWINDSQ(MMIX,MDIM)
       REAL*8        SFI(MMIX), SF(MMIX), WSP(MMIX)
       REAL*8        IFDMS, RXOLD(MDIM,0:N,MNP)

       INTEGER*8     BIN, MAXBIN, MAXBIN2, ICALCG
       INTEGER*8     HIST11(MBIN),HIST12(MBIN),HIST22(MBIN)
       REAL*8        CONST, RLOWER, RUPPER, TSTEP
       REAL*8        NIDEAL11, NIDEAL12, NIDEAL22
       REAL*8        GR11, GR12, GR22, CO(MMIX)
       REAL*8        CONST11,CONST12,CONST21,CONST22

       CHARACTER     CNFILE*30, POTK*10

       LOGICAL       OVRLAP, IFB, IFDISP, IFBRIDGE

C       PI = ACOS(-1.d0)
       OPEN (UNIT = 11, FILE = 'rdf11.dat', STATUS ='UNKNOWN')
       OPEN (UNIT = 12, FILE = 'rdf12.dat', STATUS ='UNKNOWN')
       OPEN (UNIT = 13, FILE = 'rdf22.dat', STATUS ='UNKNOWN')
       OPEN (UNIT = 14, FILE = 'rdfto.dat', STATUS ='UNKNOWN')
       OPEN (UNIT = 21, FILE = 'potep.dat' , STATUS ='UNKNOWN')
       OPEN (UNIT = 22, FILE = 'potpp.dat' , STATUS ='UNKNOWN')
       OPEN (UNIT = 23, FILE = 'potee.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-mix-tab.in', STATUS='UNKNOWN')
       READ (10,*) I
       READ (10,*) DIM, NMIX
       READ (10,*) I
       READ (10,*) SEED
       READ (10,*) I
       READ (10,*) IFB
       READ (10,*) I
       READ (10,*) (NPM(J),J=1,NMIX)
       READ (10,*) I
       READ (10,'(A)') POTK
       READ (10,*) I
       READ (10,*) FTN0
       READ (10,*) I
       READ (10,*) (FTM(J),J=1,NMIX)
       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,*) DENS
       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(I, J),I=1,NMIX),J=1,NMIX)
      READ (10,*) I
      READ (10,*) EPSR
      READ (10,*) I
      READ (10,*) ((EPSA(I, J),I=1,NMIX),J=1,NMIX)
      READ (10,*) I
      READ (10,*) EPS
      READ (10,*) I
      READ (10,*) IFDISP
      READ (10,*) I
      READ (10,*) IFBRIDGE
      READ (10,*) I
      READ (10,*) DRT
      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 AND MIXTURE DIMENSIONS  '')'')
      READ (*,*) DIM, NMIX
      WRITE(*,'('' ENTER SEED FOR RANDOM SEQUENCE            '')'')
      READ (*,*) SEED
      WRITE(*,'('' IF BOSE (T/F)                             '')'')
      READ (*,*) IFB
      WRITE(*,'('' ENTER THE NUMBER OF PARTICLES < MNP       '')'')
      READ (*,*) (NPM(J),J=1,NMIX)
      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(J),J=1,NMIX)
      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 IF INITIALIZATION NEEDED          '')'')
      READ (*,*) INIT
      WRITE(*,'('' ENTER THE DENSITY                         '')'')
      READ (*,*) DENS
      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(I, J),I=1,NMIX),J=1,NMIX)
      WRITE(*,'('' ENTER EPSR                                '')'')
      READ (*,*) EPSR
      WRITE(*,'('' ENTER EPSA                                '')'')
      READ (*,*) ((EPSA(I, J),I=1,NMIX),J=1,NMIX)
      WRITE(*,'('' ENTER EPS                                 '')'')
      READ (*,*) EPS
      WRITE(*,'('' IF DISPLACEMENT MOVE (T/F)                '')'')
      READ (*,*) IFDISP
      WRITE(*,'('' IF BRIDGE MOVE (T/F)                      '')'')
      READ (*,*) IFBRIDGE
      WRITE(*,'('' r SPACING IN THE TABLES                   '')'')
      READ (*,*) DRT
      WRITE(*,'('' IN DIMENSION (1,2,3,...)''I12/)') DIM
      WRITE(*,'('' ***********************************      ''/)'')

 1111 CONTINUE

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

      NP =  0
      DO I=1, NMIX
         NP = NP + NPM(I)
      ENDDO
      DO I=1, NMIX
         CO(I)=DBLE(NPM(I))/DBLE(NP)
      ENDDO

C     ** INITIALIZE RANDOM NUMBER GENERATOR **

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

C     ** INVERSE TEMPERATURE BETA **

      BETA = (1.d0/TEMP)

C     ** IMAGINARY TIMESTEP **

      LS  = BETA/FTNO      ! time step
      DO I=1,NMIX
         STD(I) = (LS/FTM(I))**0.5  ! standatd deviation of free-particle rho
      ENDDO

C     ** CONVERT INPUT DATA TO PROGRAM UNITS **

      SIGMA  = ( DBLE ( NP ) / DENS ) ** ( 1.0 / DIM )
      DRMAX  = ALPHA * SIGMA

C    ** WRITE INPUT DATA **

      WRITE(*,'('' SEED                      '',I10   )'') SEED
      WRITE(*,'('' POTENTIAL                 '',A     )'') POTK
      WRITE(*,'('' DIMENSIONS                '',I10   )'') DIM
      WRITE(*,'('' NUMBER OF PARTICLES       '',I10   )'') NPM
      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   )'') FTNO
      WRITE(*,'('' TIME STEP                 '',E10.4 )'') LS
      WRITE(*,'('' BOX SIDE                  '',E10.4 )'') SIGMA

C     ** MAKE POTENTIAL TABLE **
      IF (POTK .EQ. 'PCDM' .OR. POTK .EQ. 'PCDMT') THEN
         CALL PCDMT ()
      ENDIF

      IF (POTK .EQ. 'LJ' .AND. NMIX .EQ. 1) THEN
         DENSLJ = DENS * SIG(1,1) ** DBLE( DIM )
         RCUT = SIGMA/2.d0/SIG(1,1)
      ENDIF

      MAXBIN  = INT ( SIGMA/DELR )
      MAXBIN2 = INT ( MAXBIN/2.0 )
      HIST11  = 0.
      HIST12  = 0.
      HIST22  = 0.

      IF (DIM .eq. 3) THEN
         CONST11 = 4.d0 * PI * DENS * CO(1)/ 3.d0
         CONST12 = 4.d0 * PI * DENS * CO(2)/ 3.d0
         CONST21 = 4.d0 * PI * DENS * CO(1)/ 3.d0
         CONST22 = 4.d0 * PI * DENS * CO(2)/ 3.d0
      ENDIF
      IF (DIM .eq. 2) THEN
         CONST11 = PI * DENS * CO(1)
         CONST12 = PI * DENS * CO(2)
         CONST21 = PI * DENS * CO(1)
         CONST22 = PI * DENS * CO(2)
      ENDIF
      IF (DIM .eq. 1) THEN
         CONST11 = 2.d0 * DENS * CO(1)
         CONST12 = 2.d0 * DENS * CO(2)
         CONST21 = 2.d0 * DENS * CO(1)
         CONST22 = 2.d0 * DENS * CO(2)
      ENDIF

C     ** INITIALIZE CONFIGURATION **

      DO I = 1, NP
         NEXT(I) = I
         LEXT(I) = I
         DO J = 1, FTNO
            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
      ACWINDSQ= 0.0
      ACM     = 0.0
      ACATMA  = 0.0
      ACATMAB = 0.0
      ACATMAS = 0.0

C     ** READ INITIAL CONFIGURATION **
      IF ( INIT .EQ. 0 ) THEN
         PRINT*,"PATHS INITIALIZATION ..............."
         OPEN (UNIT=9, STATUS='UNKNOWN')
         CALL INITCN ( CNFILE ) ! random configuration
         CALL READCN ( CNFILE ) ! random configuration
         NS = NSTEP
      ELSEIF ( INIT .EQ. 1) THEN
         PRINT*,"RESTART FROM CONFIG  ..............."
         OPEN (UNIT=9, STATUS='UNKNOWN')
         CALL READCN ( CNFILE )
         NS = NSTEP
      ELSEIF ( INIT .EQ. 2) THEN
         PRINT*,"RESTART FULL         ..............."
         OPEN (UNIT=9, ACCESS='APPEND')
         CALL READCN ( CNFILE )
         ACATMA  = 0.0
         ACATMAB = 0.0
         ACATMAS = 0.0
         READ(20,*)(HIST11(I),I=1,MAXBIN),
```

```
     &              (HIST12(I),I=1,MAXBIN),
     &              (HIST22(I),I=1,MAXBIN),
     &          ACV,ACVSQ,ACW,ACWSQ,ACKE,ACKESQ,ACWINDSQ,ACM,NS
           NS=NS+NSTEP
        ELSE
           WRITE (*,*) 'INIT different from 0,1,2'
           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(FTNO*RANF(DUMMY))+1    ! select a random timeslice
           RS = INT(FTNO*RANF(DUMMY))+1    ! select a random referenceslice
           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
           IF (ISP(IP).NE.ISP(KP)) GOTO 77 ! different species
C     ****************************************************************
C     ** BRIDGE&SWAP MOVE (BOSE STATISTICS)                         **
C     ****************************************************************
           mcm = cp+mbm-floor((cp+mbm-.1)/ftn0)*ftn0

           rxold=rx

           nip=next(ip)
           nkp=next(kp)
           ps=0.d0

           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 kkknergy ( rx(:,i,k), k, i,
     :                   kkk )
                    keold=keold+kkk
                 enddo
              endif
           enddo

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

           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 pppnergy ( potk, rxp(:,ll), ip,
     :                   rxp(:,ll), kp, i,
     :                   vvv, www )
                 else
                    call pppnergy ( 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 pppnergy ( potk, rxp(:,ll), ip,
     :                   rxp(:,ll), kp, i,
     :                   vvv, www )
                 else
                    call pppnergy ( 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 pppnergy ( potk, rxp(:,ll), nip,
     :                   rxp(:,ll), nkp, i,
     :                   vvv, www )
                 else
                    call pppnergy ( potk, rxpp(:,ll), nip,
     :                   rxp(:,ll), nkp, i,
     :                   vvv, www )
                 endif
                 vnew=vnew+vvv
                 wnew=wnew+www
              enddo
           endif

           ps=ps+vnew-vold

           call acc_p(ps,ip,kp,cp)
           if (ps.gt.ranf(dummy)) then

              acatmab = acatmab + 1.d0
              call update(cp,rxp,ip,nkp)
              if (ip.ne.kp) then
                 call update(cp,rxpp,kp,nip)
                 call swap(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     restricted path integral
              do i=1,ftn0
                 call ifdm(isp(ip),rs,i,ifdms)
                 if ( ifdms .lt. 0. ) goto 1234
              enddo

              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 kkknergy ( rx(:,i,k), k, i,
     :                      kkk )
                       kenew=kenew+kkk
                    enddo
                 endif
              enddo
              ke=ke+kenew-keold

              v=v+vnew-vold
              w=w+wnew-wold
           endif
           goto 1235
 1234      continue
           rx=rxold
           acatmab = acatmab - 1.0
           if (ip.ne.kp) then
              acatmas = acatmas - 1.0
              call switch(next(ip),next(kp))
              call switch(mext(cp,ip),mext(cp,kp))
           endif
```

```
 1235      continue

c     build the permutations cycles in lext

          call permcyc()

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

C     ** CALCULATE INSTANTANEOUS VALUES **

                IF (POTK .EQ. 'LJ') THEN
                   VN = ( V + VLRC )
                ELSE
                   VN = V
                   WN = W
                ENDIF
                CALL CWIND( ISP(IP), WIND )

C     ** ACCUMULATE AVERAGES **

                ACV    = ACV    + VN
                ACVSQ  = ACVSQ  + VN*VN
                ACW    = ACW    + WN
                ACWSQ  = ACWSQ  + WN*WN
                ACKE   = ACKE   + KE
                ACKESQ = ACKESQ + KE*KE
                DO J=1,NMIX
                   DO I=1,DIM
                      ACWINDSQ(J,I) = ACWINDSQ(J,I) + WIND(J,I)**2.
                   ENDDO
                ENDDO
             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, FTNO

             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
c                   RXINEW(IDIM) = RXINEW(IDIM) -
c     :                  DNINT ( RXINEW(IDIM)/SIGMA )*SIGMA
                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

C     RESTRICTED PATH INTEGRAL
                   DO I=1,FTNO
                      CALL IFDM(ISP(IP),RS,I,IFDMS)
                      IF ( IFDMS .LT. 0. ) GOTO 1236
                   ENDDO
                   GOTO 1237
 1236              CONTINUE
                   V          = V  - DELTV
                   W          = W  - DELTW
                   KE         = KE - DELTKE
                   ACATMA = ACATMA - 1.0

                   DO IDIM = 1, DIM
                      RX(IDIM,C1,IP)  = RXIOLD(IDIM)
                      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
 1237              CONTINUE
                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, DENS, MAXBIN, HIST11, HIST12, HIST22)
                ENDIF

                IF (POTK .EQ. 'LJ') THEN
                   VN = ( V + VLRC )
                ELSE
                   VN = V
                   WN = W
                ENDIF
                CALL CWIND( ISP(IP), WIND )

C     ** ACCUMULATE AVERAGES **

                ACV    = ACV    + VN
                ACVSQ  = ACVSQ  + VN*VN
                ACW    = ACW    + WN
                ACWSQ  = ACWSQ  + WN*WN
                ACKE   = ACKE   + KE
                ACKESQ = ACKESQ + KE*KE
                DO J=1,NMIX
                   DO I=1,DIM
                      ACWINDSQ(J,I) = ACWINDSQ(J,I) + WIND(J,I)**2.
                   ENDDO
                ENDDO
             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 **

          SF  = 0.0
          SFI = 0.0
          DO I=1, NMIX
             DO IDIM = 1, DIM
                SFI(I) = SFI(I) + WIND(I,IDIM)*WIND(I,IDIM)
                SF(I)  = SF(I)  + ACWINDSQ(I,IDIM)
             ENDDO
          ENDDO
          DO I=1, NMIX
             SFI(I) = SFI(I)*FTM(I)/(DIM*BETA*NPM(I))
             SF(I)  = SF(I)*FTM(I)/(DIM*BETA*NPM(I)*ACM)
          ENDDO
          IF (ISP(IP).EQ.1)THEN
             WSP(1)=1
             WSP(2)=0
          ELSE
             WSP(1)=0
             WSP(2)=1
          ENDIF

          CALL FLUSH(9)
          IF ( MOD ( STEP, IPRINT ) .EQ. 0 ) THEN
             WRITE(9,*) STEP,DIM*NP/2/LS-KE/FTNO,V/FTNO,W/FTNO,
     :                  SFI,WSP
C             WRITE(9,*) STEP, DIM*NP/2/LS-KE/FTNO, V/FTNO, W/FTNO
          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 ( FTN0 * 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/FTN0, V/FTN0, W/FTN0, SFI
              WRITE(*,*) '          <ke>            <v>            <w>
     :         <sf>'
              WRITE(*,'(4(2XE13.7))') ACKE/ACM/FTN0, ACV/ACM/FTN0,
     :                                 ACW/ACM/FTN0, SF

           ENDIF

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

C     ** WRITE OUT THE CONFIGURATION AT INTERVALS **

              CALL WRITCN ( CNFILE )
              REWIND(20)
              CALL FLUSH(20)
              WRITE(20,*)(HIST11(I),I=1,MAXBIN),
     &                   (HIST12(I),I=1,MAXBIN),
     &                   (HIST22(I),I=1,MAXBIN),
     &              ACV,ACVSQ,ACW,ACWSQ,ACKE,ACKESQ,ACWINDSQ,ACM,STEP

C     ** WRITE OUT THE g(r) **
              IF ( STEP .GT. IEQUI ) THEN
                 TSTEP = DBLE((STEP - IEQUI)*INT(FTN0/ICALCG))
                 REWIND(11)
                 REWIND(12)
                 REWIND(13)
                 REWIND(14)
                 DO BIN = 1, MAXBIN2
                    RLOWER = DBLE ( BIN - 1 ) * DELR
                    RUPPER = RLOWER + DELR
                    NIDEAL11 = CONST11*( RUPPER**DIM - RLOWER**DIM )
                    NIDEAL12 = CONST12*( RUPPER**DIM - RLOWER**DIM )
                    NIDEAL22 = CONST22*( RUPPER**DIM - RLOWER**DIM )
                    GR11 = DBLE(HIST11(BIN))/TSTEP/DBLE(NPM(1))/NIDEAL11
                    GR12 = DBLE(HIST12(BIN))/TSTEP/DBLE(NPM(1))/NIDEAL12
                    GR22 = DBLE(HIST22(BIN))/TSTEP/DBLE(NPM(2))/NIDEAL22
                    CALL FLUSH(11)
                    CALL FLUSH(12)
                    CALL FLUSH(13)
                    CALL FLUSH(14)
                    WRITE (11,*) DBLE(BIN-1)*DELR, GR11
                    WRITE (12,*) DBLE(BIN-1)*DELR, GR12
                    WRITE (13,*) DBLE(BIN-1)*DELR, GR22
                    WRITE (14,*) DBLE(BIN-1)*DELR,
     :              C0(1)**2*GR11+2.d0*C0(1)*C0(2)*GR12+C0(2)**2*GR22
                 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 ''/)')

        CLOSE (UNIT = 9 )
        CLOSE (UNIT = 11)
        CLOSE (UNIT = 12)
        CLOSE (UNIT = 13)
        CLOSE (UNIT = 14)
        CLOSE (UNIT = 21)
        CLOSE (UNIT = 22)
        CLOSE (UNIT = 23)
C     ****************************************************************
C     ****************************************************************
C     ****************************************************************
C     ** THE END                                                    **
C     ****************************************************************
C     ****************************************************************
C     ****************************************************************
        STOP
        END


        subroutine permcyc()
        implicit none
!     given a particle i lext(i) returns the cycle it belongs to
        INCLUDE       'mc-mix-tab.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 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(p,ip,kp,j)
        implicit none
!     complete acceptance probability
        INCLUDE       'mc-mix-tab.par'

        integer*8 ip,kp,j,idim
        real*8 p, rho
        real*8 rxink,rxnik
        real*8 rxini,rxknk
        integer*8 mcm

        p=exp(-ls*p)         ! contribution from the pair potential
        if (ip.eq.kp) return

        mcm = j+mbm-floor((j+mbm-.1)/ftn0)*ftn0
        rho=0.d0
        do idim=1,dim
           if(j+mbm.le.ftn0)then
              rxink=rx(idim,j,ip)-rx(idim,mcm,kp)
              rxnik=rx(idim,j,kp)-rx(idim,mcm,ip)
              rxini=rx(idim,j,ip)-rx(idim,mcm,ip)
              rxknk=rx(idim,j,kp)-rx(idim,mcm,kp)
              rxink=rxink-dnint(rxink/sigma)*sigma
              rxnik=rxnik-dnint(rxnik/sigma)*sigma
```

```
            rxini=rxini-dnint(rxini/sigma)*sigma
            rxknk=rxknk-dnint(rxknk/sigma)*sigma
         else
            rxink=rx(idim,j,ip)-rx(idim,mcm,next(kp))
            rxnik=rx(idim,j,kp)-rx(idim,mcm,next(ip))
            rxini=rx(idim,j,ip)-rx(idim,mcm,next(ip))
            rxknk=rx(idim,j,kp)-rx(idim,mcm,next(kp))
            rxink=rxink-dnint(rxink/sigma)*sigma
            rxnik=rxnik-dnint(rxnik/sigma)*sigma
            rxini=rxini-dnint(rxini/sigma)*sigma
            rxknk=rxknk-dnint(rxknk/sigma)*sigma
         endif
         rho=rho+rxink**2+rxnik**2-rxini**2-rxknk**2
      enddo
      rho=ftm(isp(ip))*rho/(2.*mbm*ls)
      p=p*exp(-rho)
      return
      end


      subroutine update(j,rxp,ip,nip)
      implicit none
!     updates a portion of the current path x using the proposed path xp
      INCLUDE       'mc-mix-tab.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
               rx(idim,k,ip)=rxp(idim,l)
            enddo
         enddo
      else
         do k=j+1,ftn0
            l=l+1
            do idim=1,dim
               rx(idim,k,ip)=rxp(idim,l)
            enddo
         enddo
         do k=1,mcm-1
            l=l+1
            do idim=1,dim
               rx(idim,k,nip)=rxp(idim,l)
            enddo
         enddo
      endif
      do idim=1,dim
         rx(idim,ftn0+1,ip)=rx(idim,1,nip)
         rx(idim,0,nip)=rx(idim,ftn0,ip)
      enddo

      return
      end


      subroutine swap(j,ip,nip,kp,nkp)
      implicit none
!     updates a portion of the current path x using the proposed path xp
      INCLUDE       'mc-mix-tab.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=rx(idim,k,ip)
               rx(idim,k,ip)=rx(idim,k,kp)
               rx(idim,k,kp)=rr
            enddo
         enddo
         do idim=1,dim
            rx(idim,ftn0+1,ip)=rx(idim,1,nkp)
            rx(idim,ftn0+1,kp)=rx(idim,1,nip)
            rx(idim,0,nip)=rx(idim,ftn0,kp)
            rx(idim,0,nkp)=rx(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-mix-tab.par'

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

      out=0
      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))/sigma)*sigma)
      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/sigma)*sigma
            xnew(idim,j)=xnew(idim,l1)+d/dble(l3-l1)+xi(s)
c            if (xnew(idim,j).gt.sigma/2.or.
c     :           xnew(idim,j).lt.-sigma/2) then
c               out=1
c               return
c            endif
c            xnew(idim,j)=xnew(idim,j)-dnint(xnew(idim,j)/sigma)*sigma
         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-mix-tab.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,FTN0
            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.D0
               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-mix-tab.par'

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

      REAL*8        RXII, RXIJ
      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, FTN0
         DO 100 I = 1, NP - 1
            DO 99 J = I + 1, NP
               RIJSQ = 0.d0
               DO IDIM = 1, DIM
                  RXIJ = RX(IDIM,TAU,I) - RX(IDIM,TAU,J)
C     ** MINIMUM IMAGE THE PAIR SEPARATIONS **
                  RXIJ = RXIJ -
     :                 DNINT ( RXIJ/SIGMA )*SIGMA
                  RIJSQ = RIJSQ + RXIJ * RXIJ
               ENDDO

               CALL POT (I, J, RIJSQ, VIJ, WIJ, POTK)
               VV   = VV + VIJ
               WW   = WW + WIJ
```

```
 99          CONTINUE
 100      CONTINUE
          V=VV
          W=WW
       ENDDO

C     KINETIC ACTION

       KK     = 0.0

       DO TAU = 1, FTNO
          DO I = 1, NP
             DO IDIM = 1, DIM
                RXII = RX(IDIM,TAU,I)-RX(IDIM,TAU+1,I)
                RXII = RXII -
     :               DNINT ( RXII/SIGMA )*SIGMA
                KK=KK+FTM(ISP(I))*(RXII**2.)/(2.DO*LS**2.)
             ENDDO
          ENDDO
          KE=KK
       ENDDO

       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-mix-tab.par'

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

       REAL*8        RXIJ, RIJSQ, VIJ, WIJ

       V      = 0.0
       W      = 0.0

C    ** LOOP OVER ALL MOLECULES EXCEPT I  **

       DO 100 J = 1, NP

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

                RXIJ = RXIJ -
     :               DNINT ( RXIJ/SIGMA )*SIGMA
                RIJSQ = RIJSQ + RXIJ * RXIJ
             ENDDO

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

 90       ENDIF

100    CONTINUE

       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-mix-tab.par'

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

       REAL*8        RXIJ, 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
             DO IDIM = 1, DIM
                RXIJ = RXI(IDIM) - RX(IDIM,TAU,K)

                RXIJ = RXIJ -
     :               DNINT ( RXIJ/SIGMA )*SIGMA
                RIJSQ = RIJSQ + RXIJ * RXIJ
             ENDDO

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

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

                   RXIJ = RXIJ -
     :                  DNINT ( RXIJ/SIGMA )*SIGMA
                   RIJSQ = RIJSQ + RXIJ * RXIJ
                ENDDO

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

             ENDIF

          ENDIF

 100    CONTINUE

C       RETURN

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

             RXIJ = RXIJ -
     :            DNINT ( RXIJ/SIGMA )*SIGMA
             RIJSQ = RIJSQ + RXIJ * RXIJ
          ENDDO

          CALL POT (I, J, RIJSQ, VIJ, WIJ, POTK)
          V   = V + VIJ
          W   = W + WIJ
       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-mix-tab.par'

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

       KE = 0.d0
       DO IDIM = 1, DIM
          RXIP = RX(IDIM,TAU-1,I)
          RXIS = RX(IDIM,TAU+1,I)
          RXII = RXI(IDIM) - RXIP
          RXII = RXII - DNINT ( RXII/SIGMA )*SIGMA
          KE=KE+0.5*FTM(ISP(I))*(RXII/LS)**2.
          RXII = RXI(IDIM) - RXIS
          RXII = RXII - DNINT ( RXII/SIGMA )*SIGMA
          KE=KE+0.5*FTM(ISP(I))*(RXII/LS)**2.
       ENDDO

       RETURN
       END


       SUBROUTINE KKKNERGY ( 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-mix-tab.par'

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

       KE = 0.d0
       DO IDIM = 1, DIM
          RXIS = RX(IDIM,TAU+1,I)
          RXII = RXI(IDIM) - RXIS
          RXII = RXII - DNINT ( RXII/SIGMA )*SIGMA
          KE=KE+0.5*FTM(ISP(I))*(RXII/LS)**2.
       ENDDO

       RETURN
       END


       SUBROUTINE CWIND ( ISPEC, WIND )
       IMPLICIT NONE
C    ***************************************************************
C    ** RETURNS THE WINDING NUMBER.                               **
C    ***************************************************************
       INCLUDE     'mc-mix-tab.par'

       REAL*8        WIND(MMIX,MDIM), WI, RX1, RX2
       INTEGER*8     I, J, IDIM, ISPEC

       WIND = 0.d0
       DO I = 1, NP
          IF (ISP(I).NE.ISPEC) GOTO 10
          DO IDIM = 1, DIM
             DO J = 1, FTNO - 1
                RX1 = RX(IDIM,J  ,I)
                RX2 = RX(IDIM,J+1,I)
                WI  = RX2-RX1-DNINT((RX2-RX1)/SIGMA)*SIGMA
                WIND(ISPEC,IDIM) = WIND(ISPEC,IDIM) + WI
             ENDDO
          ENDDO
 10       CONTINUE
       ENDDO

       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-mix-tab.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),I3)'

       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),
     :                             ISP(I)
 90       ENDDO
          READ ( CNUNIT,13 )
          READ ( CNUNIT,13 )
          DO IDIM = 1, DIM
             RX(IDIM,0,NI) = RX(IDIM,FTNO,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-mix-tab.par'

       CHARACTER   CNFILE*(*), MYFMT*77

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

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

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

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

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

       CLOSE ( UNIT = CNUNIT )

       RETURN
       END


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

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

       INTEGER*8   CNUNIT, I, J, IDIM, I1, IP, NNP
       REAL*8      RXI(DIM), RXII(DIM, NP), RXIJ, RIJSQ
       PARAMETER ( CNUNIT = 10 )

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

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

       WRITE ( CNUNIT,* ) '#',FTNO,NP
       RXII = 0.d0

       IP = 1
       NNP = NPM(IP)
       DO I = 1, NP
          ISP(I)=IP
          IF (I.GE.NNP) THEN
             IP = IP + 1
             NNP = NNP + NPM(IP)
          ENDIF
       ENDDO

       DO I = 1, NP
          WRITE ( CNUNIT,* ) '#', NEXT(I)
 10       DO IDIM = 1, DIM
             RXI(IDIM) = SIGMA*(RANF(0.d0)-.5)
          ENDDO
          DO I1 = 1, NP
             RIJSQ = 0.d0
             DO IDIM = 1, DIM
                RXIJ = RXI(IDIM)-RXII(IDIM,I1)
```

```
                RXIJ = RXIJ  -
     :               DNINT ( RXIJ/SIGMA )*SIGMA
                RIJSQ = RIJSQ + RXIJ * RXIJ
              ENDDO
              IF (RIJSQ.LE.EPSA(ISP(I),ISP(I1))**2) THEN
                 GOTO 10
              ENDIF
           ENDDO
           DO IDIM = 1, DIM
              RXII(IDIM,I) = RXI(IDIM)
           ENDDO
           DO J = 1, FTN0+1
              WRITE ( CNUNIT, MYFMT ) I, (RXI(IDIM),IDIM=1,DIM), ISP(I)
              DO IDIM = 1, DIM
                 RX(IDIM,J,I) = RXI(IDIM)
              ENDDO
           ENDDO
           WRITE ( CNUNIT,13 ) ''
           WRITE ( CNUNIT,13 ) ''
        ENDDO
 13     FORMAT(A2)

        CLOSE ( UNIT = CNUNIT )

        RETURN
        END



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

        INTEGER*8 I, J, K, IT, ITO
        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' .AND. NMIX .EQ. 1) THEN
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C     W. L. McMillan, Phys. Rev. A 138, 442 (1965)     C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C           RMIN = TINY(PI)
           RMIN = 0.d0
           RIJSQ = RIJSQ / (SIG(I,J) * SIG(I,J))
           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(I,J)) THEN
              VIJ = EPSR
           ENDIF
        ELSEIF (POTK .EQ. 'PSW') THEN
           RIJ = SQRT( RIJSQ )
           IF (RIJ .LE. SIG(I,J)) THEN
              VIJ = EPSR
           ELSEIF (RIJ .LE. SIG(I,J)+RCUT .AND. RIJ .GT. SIG(I,J)) THEN
              VIJ = -EPSA(I,J)
           ENDIF
        ELSEIF (POTK .EQ. 'COULOMBHS') THEN
c           F=0.
c           DO K=1,NMIX
c              F = F+SIG(1,K)*NPM(K)/(NPM(K)-1)
c           ENDDO
           RIJ = SQRT( RIJSQ )
           IF (RIJ .GT. EPSA(ISP(I),ISP(J))) THEN
              IF (DIM .EQ. 3) THEN
c                 VIJ = -F*COU3*RCUT/SIGMA/2.
c                 VIJ = VIJ + RCUT*/RIJ
                 VIJ = SIG(ISP(I),ISP(J))*RCUT/RIJ
              ELSEIF (DIM .EQ. 2) THEN
c                 VIJ = -F*(6.-PI+LOG(4.)-4.*LOG(SIGMA/RCUT))/4.
c                 VIJ = VIJ - LOG(RIJ/RCUT)
                 VIJ = -SIG(ISP(I),ISP(J))*LOG(RIJ/RCUT)
              ELSEIF (DIM .EQ. 1) THEN
c                 VIJ = F*SIGMA/RCUT/4.
c                 VIJ = VIJ - RIJ/RCUT
                 VIJ = -SIG(ISP(I),ISP(J))*RIJ/RCUT
              ENDIF
           ELSE
              VIJ = 10.**10.
           ENDIF
        ELSEIF (POTK .EQ. 'COULOMB') THEN
c           F=0.
c           DO K=1,NMIX
c              F = F+SIG(1,K)*NPM(K)/(NPM(K)-1)
c           ENDDO
           RIJ = SQRT( RIJSQ )
           IF (RIJ .GT. EPSA(ISP(I),ISP(J))) THEN
              IF (DIM .EQ. 3) THEN
c                 VIJ = -F*COU3*RCUT/SIGMA/2.
c                 VIJ = VIJ + RCUT*/RIJ
                 VIJ = SIG(ISP(I),ISP(J))*RCUT/RIJ
              ELSEIF (DIM .EQ. 2) THEN
c                 VIJ = -F*(6.-PI+LOG(4.)-4.*LOG(SIGMA/RCUT))/4.
c                 VIJ = VIJ - LOG(RIJ/RCUT)
                 VIJ = -SIG(ISP(I),ISP(J))*LOG(RIJ/RCUT)
              ELSEIF (DIM .EQ. 1) THEN
c                 VIJ = F*SIGMA/RCUT/4.
c                 VIJ = VIJ - RIJ/RCUT
                 VIJ = -SIG(ISP(I),ISP(J))*RIJ/RCUT
              ENDIF
           ELSE
              RIJ = EPSA(ISP(I),ISP(J))
              IF (DIM .EQ. 3) THEN
c                 VIJ = -F*COU3*RCUT/SIGMA/2.
c                 VIJ = VIJ + RCUT*/RIJ
                 VIJ = SIG(ISP(I),ISP(J))*RCUT/RIJ
              ELSEIF (DIM .EQ. 2) THEN
c                 VIJ = -F*(6.-PI+LOG(4.)-4.*LOG(SIGMA/RCUT))/4.
c                 VIJ = VIJ - LOG(RIJ/RCUT)
                 VIJ = -SIG(ISP(I),ISP(J))*LOG(RIJ/RCUT)
              ELSEIF (DIM .EQ. 1) THEN
c                 VIJ = F*SIGMA/RCUT/4.
c                 VIJ = VIJ - RIJ/RCUT
                 VIJ = -SIG(ISP(I),ISP(J))*RIJ/RCUT
              ENDIF
           ENDIF
        ELSEIF (POTK .EQ. 'PCDM') THEN
           IF (DIM .NE. 3) THEN
              WRITE (*,*) 'DIM is not 3 in PCDM !'
              STOP
           ENDIF
           RIJ = SQRT( RIJSQ )
           ITO = INT(RIJ/DRT)
           IF (ISP(I) .NE. ISP(J)) THEN
              vij = potep(it0)+(potep(it0+1)-potep(it0))*
     :             (rij-rt(it0))/drt
           ELSE
              VIJ = SIG(ISP(I),ISP(J))*RCUT/RIJ
           ENDIF
        ELSEIF (POTK .EQ. 'PCDMT') THEN
           IF (DIM .NE. 3) THEN
              WRITE (*,*) 'DIM is not 3 in PCDM !'
              STOP
           ENDIF
           RIJ = SQRT( RIJSQ )
           ITO = INT(RIJ/DRT)
           IF (ISP(I) .NE. ISP(J)) THEN
              vij = potep(it0)+(potep(it0+1)-potep(it0))*
     :             (rij-rt(it0))/drt
           ELSE
              IF (ISP(I) .EQ. 1) THEN
                 vij = potee(it0)+(potee(it0+1)-potee(it0))*
     :                (rij-rt(it0))/drt
              ELSE
                 vij = potpp(it0)+(potpp(it0+1)-potpp(it0))*
     :                (rij-rt(it0))/drt
              ENDIF
           ENDIF
        ELSEIF (POTK .EQ. 'QHS') THEN
c           F=0.
c           DO K=1,NMIX
c              F = F+SIG(1,K)*NPM(K)/(NPM(K)-1)
c           ENDDO
           IF (DIM .EQ. 3) THEN
              IF (RIJSQ .LE. SIG(I,J)*SIG(I,J)) THEN
                 VIJ = 10.**10.
              ELSE
c                 VIJ = -F*QHS3*(SIGMA-SIG(I,J))/SIGMA**3/2./FTM(I)
                 VIJ = VIJ + (2.*RIJSQ+SIG(I,J)**2.)/
     :                (RIJSQ-SIG(I,J)**2.)**2./2./FTM(I)
                 WIJ = -2.*RIJSQ*(RIJSQ+2.*SIG(I,J)**2.)/
     :                (RIJSQ-SIG(I,J)**2.)**3./FTM(I)
              ENDIF
           ELSE
              WRITE(*,*) 'QHS in dimension different from 3'
              STOP
           ENDIF
        ELSEIF (POTK .EQ. 'HARMONIC') THEN
           VIJ = 0.5*RIJSQ/RCUT**2.
        ELSE
           WRITE(*,*) 'NO PAIR POTENTIAL AVAILABLE!'
           STOP
        ENDIF

        RETURN
        END



        SUBROUTINE GOFR ( C, DENS, MAXBIN, HIST11, HIST12, HIST22 )
        IMPLICIT NONE
C    *****************************************************************
C    ** CALCULATES THE RADIAL DISTRIBUTION FUNCTION HISTOGRAM.      **
C    *****************************************************************
        INCLUDE     'mc-mix-tab.par'

        REAL*8        RXIJ, RIJSQ, RIJ, DENS
        INTEGER*8     C, IDIM, I, J, BIN, MAXBIN
        INTEGER*8     HIST11(MAXBIN), HIST12(MAXBIN), HIST22(MAXBIN)

        DO I = 1, NP - 1
```

```
            DO J = I + 1, NP
               RIJSQ = 0.
               DO IDIM = 1, DIM
                  RXIJ  = RX(IDIM,C,I) - RX(IDIM,C,J)
                  RXIJ  = RXIJ - ANINT ( RXIJ/SIGMA ) * SIGMA
                  RIJSQ = RIJSQ + RXIJ * RXIJ
               ENDDO
C     CALCULATE HISTOGRAM FOR g(r)
               RIJ = SQRT ( RIJSQ )
               BIN = INT ( RIJ/DELR ) + 1

               IF (BIN .LE. MAXBIN) THEN
                  IF (ISP(I) .EQ. 1 .AND. ISP(J) .EQ. 1) THEN
                     HIST11(BIN) = HIST11(BIN) + 2
                  ELSEIF(ISP(I) .EQ. 2 .AND. ISP(J) .EQ. 2) THEN
                     HIST22(BIN) = HIST22(BIN) + 2
                  ELSEIF(ISP(I) .NE. ISP(J)) THEN
                     HIST12(BIN) = HIST12(BIN) + 1
                  ENDIF
               ENDIF

            ENDDO
         ENDDO

         RETURN
         END


ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
c     determines the sign of the free spinless Fermi density matrix
c     dim  = spatial dimensions
c     nf   = number of fermions
c     beta = inverse temperature
c     itau0= initial time slice
c     itau1= final   time slice
c     sign = sign of the free fermion density matrix
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine ifdm(ispec,itau0,itau1,sign)
      implicit none
      include 'mc-mix-tab.par'
      integer*8 ispec,itau0,itau1,i,j,ii,jj,idim,npp
      real*8 sum,DMGT,epsilon,det,sign
      real*8 aaa(np,np),beta,lambda

      epsilon=1.d-10
      beta=ls*abs(itau1-itau0)
      lambda=1./2./ftm(ispec)

      if (itau0.eq.itau1) then
         sign=1.
         return
      endif

      npp=0
      do i=1,ispec-1
         npp=npp+npm(isp(i))
      enddo
c     set up the free particle density matrix
      do i=1,npm(ispec)
         do j=1,npm(ispec)
            ii=i+npp
            jj=j+npp
            sum=0.d0
            do idim=1,dim
               sum=sum+(rx(idim,itau0,jj)-rx(idim,itau1,ii))**2.
            enddo
            sum=sum/(4*beta*lambda)
            aaa(i,j)=exp(-sum)
            aaa(i,j)=aaa(i,j)/(4*pi*beta*lambda)**(dim/2.)
         enddo
      enddo

      det=DMGT(epsilon,npm(ispec),aaa)

c     sign of the determinant
      sign=det

      return
      end


!The subroutine TSRGT applies to input real square matrix A(n,n) the upper
!triangularization algorithm of Gauss method with full pivoting and keeps
!trace of successive transformations done in integer vectors KP and LP.
!-----------------------------------------------------------------------
!  Input parameters:
!  eps        precision (real*8)
!  n          size of A matrix (integer)
!  A          pointer to input real square matrix (real*8)
!  Output parameters:
!  it         flag=1 if A matrix ok, =0 if A matrix is singular (integer)
!  C          pointer to table storing main diagonal elements and supra-
!             diagonal elements of upper triangular matrix and the multi-
!             plying coefficients used during triangularization process
!  KP         table storing informations concerning the column exchanges
!             during process (integer)
!  LP         table storing informations concerning the line exchanges
!             during process (integer)
!-----------------------------------------------------------------------
!The table C is first initialized to A matrix, then receives at each step k
!of the triangularization process, usefull elements of A matrix at step k for
!k=1,2,...n.
!The variables po(real*8), lo and ko(integer) store respectively pivot at step k,
!its line number and its column number.
!-----------------------------------------------------------------------
      Subroutine TSRGT(eps, n, A, it, C, Kp, Lp)
      real*8 eps
      integer*8 n,it
      real*8 A(n,n), C(n,n)
      integer*8 Kp(n),Lp(n)
      real*8  po,t0
      C=A
      it=1
      k=1
      do while (it==1.and.k<n)
         po=C(k,k)
         lo=k
         ko=k
         do i=k, n
            do j=k, n
               if (dabs(C(i,j))>dabs(po)) then
                  po=C(i,j)
                  lo=i
                  ko=j
               end if
            end do
         end do
         Lp(k)=lo
         Kp(k)=ko
         if (dabs(po)<eps) then
            it=0
         else
            if (lo.ne.k) then
               do j=k, n
                  t0=C(k,j)
                  C(k,j)=C(lo,j)
                  C(lo,j)=t0
               end do
            end if
            if (ko.ne.k) then
               do i=1, n
                  t0=C(i,k)
                  C(i,k)=C(i,ko)
                  C(i,ko)=t0
               end do
            end if
            do i=k+1, n
               C(i,k)=C(i,k)/po
               do j=k+1, n
                  C(i,j)=C(i,j)-C(i,k)*C(k,j)
               end do
            end do
            k=k+1
         end if
      end do
      if (it==1.and.dabs(C(n,n))<eps)  it=0
      return
      End                       !TSRGT


!The function DMGT returns the determinant of a real square matrix
!A(n,n) by Gauss method with full pivoting.
!-----------------------------------------------------------------------
!  Input parameters:
!  eps        precision (real*8)
!  n          size of A matrix (integer)
!  A          pointer to input real square matrix
!  Output parameters:
!  None
!-----------------------------------------------------------------------
!The procedure TSRGT is used to reduce A matrix to an upper triangular matrix.
!Output variables are it(integer), C(n,n), Kp(n) and Lp(n).
!If it=0, matrix A is singular, if it=1, matrix A is regular. Table C contains
!at location i,j (j>=i) the corresponding element of the upper triangular matrix.
!Tables Lp and Kp contain informations relative to exchanges of line or column
!that occured during the process. For instance, the element number k of Lp is
!an integer <> k if an exchange of line has been made at step k (k=1,2,...,n).
!The number of exchanges of lines and columns is stored in integer L. the
!determinant of A matrix is stored in d0 (real*8).
!-----------------------------------------------------------------------
      real*8 Function DMGT(eps, n, A)
      integer*8 n, it, Kp(n), Lp(n)
      real*8 eps, A(n,n), C(n,n)
      real*8 d0

      call TSRGT(eps,n,A,it,C,Kp,Lp) !call triangularization subroutine
      if (it==0) then
         d0=0.d0                !matrix singular, det=0
      else                      !matrix regular, det<>0
         d0=1.d0
         do k=1, n
  d0=d0*C(k,k)
       end do
       l=0
       do k=1, n-1
          if (Lp(k).ne.k)  l=l+1
          if (Kp(k).ne.k)  l=l+1
       end do
      if (MOD(l,2).ne.0) d0=-d0 !l is odd
      end if
      DMGT=d0                   !return determinant
      return
      End


      subroutine pcdmt()
c*****************************************************************
c   program to compute the 2 particle relative density matrix for the
c   1/r potential in 3 dimensions.
c   ATOMIC UNITS
c   xkappa=1/(2*mu)
c   mu=m_1 *m_2/(m_1+m_2)
c   z=z_1 *z_2/xkappa
c   e.g. for e-e xkappa=1, z=1,  for e-p xkappa=.5, z=-2
c   for p-p xkappa=1/2000 z=2000
c*****************************************************************
      implicit integer (i-n)
      implicit real*8 (a-h,o-z)
      include       'mc-mix-tab.par'
      real timearray(2),time
c      data xkappa/0.5/pi/3.141592654/z/-2.0/           !e-p
c      data xkappa/0.0005/pi/3.141592654/z/2000.0/      !p-p
```

```
c      data xkappa/1.0/pi/3.141592654/z/1.0/            !e-e

c      call dtime(timearray,time)

       theta=0.0
       beta=ls

c      write(21,899) beta,xkappa,z,theta
c      write(*,899) beta,xkappa,z,theta
c899   format(2x,"3d",5x,"beta=",f7.4,5x,"kappa=",f7.4,5x,"Z=",f7.4,
c     $       5x,"theta=",f7.4/5x,"r",10x,"rho(r,r)",10x,"P(r)"//)

       do  ir=1,int(sigma*sqrt(2.)/drt)
          r= ir*drt
          rt(ir)=r
c ep component
          xkappa=0.5
          z=-2.0
          xint=rhocoul(r,r,theta,beta,xkappa,z)
c         xint=rhocoul(x1,x2,theta,beta,xkappa,z)
          r12sq=r*r+r*r-2.*r*r*cos(theta)
          rhofreed=exp(-r12sq/(4.*xkappa*beta))/(4.*pi*xkappa*beta)**1.5
          pofr=-dlog(abs(xint/rhofreed))
          potep(ir)=pofr
          write(21,900) r,xint,pofr
c          write(*,900) r,xint,pofr
          call flush(21)
c ee component
          xkappa=1.0
          z=1.0
          xint=rhocoul(r,r,theta,beta,xkappa,z)
c         xint=rhocoul(x1,x2,theta,beta,xkappa,z)
          r12sq=r*r+r*r-2.*r*r*cos(theta)
          rhofreed=exp(-r12sq/(4.*xkappa*beta))/(4.*pi*xkappa*beta)**1.5
          pofr=-dlog(abs(xint/rhofreed))
          potee(ir)=pofr
          write(22,900) r,xint,pofr
c          write(*,900) r,xint,pofr
          call flush(22)
c pp component
          xkappa=0.000544617
          z=1836.15
          xint=rhocoul(r,r,theta,beta,xkappa,z)
c         xint=rhocoul(x1,x2,theta,beta,xkappa,z)
          r12sq=r*r+r*r-2.*r*r*cos(theta)
          rhofreed=exp(-r12sq/(4.*xkappa*beta))/(4.*pi*xkappa*beta)**1.5
          pofr=-dlog(abs(xint/rhofreed))
          potpp(ir)=pofr
          write(23,900) r,xint,pofr
c          write(*,900) r,xint,pofr
          call flush(23)
 900      format(4x,f6.3,3x,d14.7,4x,d14.7)
       enddo

c      call dtime(timearray,time)
c      write(21,901) timearray(1),timearray(2)
c      write(*,901) timearray(1),timearray(2)
c 901  format(//" elapsed user time=",d12.5,5x," elapsed system time=",
c     $         d12.5/)
       return
       end


       function rhocoul(r1,r2,theta,beta,xkappa,z)
c*********************************************************************
c   This function returns the 2 particle relative density matrix for  *
c   the coulomb potential                                             *
c*********************************************************************
c   r1        "initial" radius/bohr radius
c   r2        "final" radius/bohr radius
c   theta     angle between r1 and r2
c   beta      inverse temperature *e**2/bohr radius
c   xkappa    hbar**2/(2*reduced mass*e**2*bohr radius)
c   z         Z1*Z2/xkappa (can be either positive or negative)
c*********************************************************************
       implicit integer (i-n)
       implicit real*8 (a-h,o-z)
       common /specs/ r11,r22,ttheta,bbeta,xkap,zz
       external tgrand

       r11=r1
       r22=r2
       ttheta=theta
       bbeta=beta
       xkap=xkappa
       zz=z
       uklim=sqrt(5.0/(beta*xkappa))
       call qromb(tgrand,0.d0,10.*uklim,xint1)
c      call qsimp(tgrand,0.0,2.*uklim,xint1)
c      call qromb(tgrand,uklim,2.*uklim,xint2)
       rhocoul=xint1
       bstuf=0.0
       if(z.lt.0.0) bstuf=bound(r1,r2,theta,beta,xkappa,z,n)
       rhocoul=rhocoul+bstuf
c      write(3,900) bstuf,n,xint1
c900   format(" bound part=",e14.7,3x,"n=",i5,3x,"continuum part=",e14.7)

       return
       end

       function bound(r1,r2,theta,beta,xkappa,z,n)
c*********************************************************************
c   returns bound state contribution to 3d coulomb density matrix     *
c   sum over principal quantum numbers is terminated when the         *
c   contribution is less than epsbd of the total accumulated bound    *
c   state sum                                                         *
c*********************************************************************
       implicit integer (i-n)
       implicit real*8 (a-h,o-z)
       data fourpii/.079577472/epsbd/1.d-7/

       sum=0.0
       r12=sqrt(r1*r1+r2*r2-2.*r1*r2*cos(theta))
       x=(r1+r2+r12)/2.
       y=(r1+r2-r12)/2.
       zabs=abs(z)

       do 1 n=1,75
       fxp=1.0
       zxn=zabs*x/float(n)
       fx=1.-zxn
       fyp=1.0
       zyn=zabs*y/float(n)

       fy=1.-zyn
       if(n.ne.1) then
     do 2 i=1,n-1
     fxn=((2.*i+1.-zxn)*fx-i*fxp)/(i+1.)
     fxp=fx
     fx=fxn
     if(x.eq.y) goto 2
     fyn=((2.*i+1.-zyn)*fy-i*fyp)/(i+1.)
     fyp=fy
     fy=fyn
 2      continue
 endif
       prefac=fourpii*exp(.25*beta*xkappa*z*z/(n*n))*(.5*zabs**3/n**5)*
      $              exp(-.5*zabs*(r1+r2)/n)
       if(x.ne.y) then
         term=(n**3/zabs)*prefac*(fxp*fy-fx*fyp)/r12
       else
     if(x.ne.0.0) then
              term=(n**2)*prefac*((n**2/(zabs*r1))*(fx-fxp)**2+fx*fxp)
           else
               term=(n**2)*prefac
           endif
       endif
       sum=sum+term
       oldbound=bound
       tail=(n-1)*(n-1)*term/float(2*n-1)
       bound=sum+tail
c      write(3,900) n,term,sum,bound
900    format(" n=",i5,2x,"term=",d15.8,3x,"partial=",d15.8,
     $        2x,"bound=",d15.8)
       if(abs(bound-oldbound).lt.epsbd*abs(bound)) return
1      continue

       return
       end

       function tgrand(wavek)
c*********************************************************************
c   integrand for continuum part of the 3d coulomb density matrix     *
c*********************************************************************
       implicit integer (i-n)
       implicit real*8 (a-h,o-z)
       dimension ffx(1),ffxd(1),ggx(1),ggxd(1)
       common /specs/ r1,r2,theta,beta,xkappa,z
       data pi/3.141592654/twopi/6.283185308/twopisq/19.73920881/
       data acura/1.d-11/step/100./

       tgrand=0.0
       if(wavek.eq.0.0) return
       r12=sqrt(r1*r1+r2*r2-2.*r1*r2*cos(theta))
       x=(r1+r2+r12)/2.
       y=(r1+r2-r12)/2.
       xk=wavek*x
       yk=wavek*y
       gibbs=exp(-beta*xkappa*wavek**2)
       fo1=z/(2.0*wavek)

       if(x.eq.y) goto 1

       call rcwfn(xk,fo1,0,0,ffx,ffxd,ggx,ggxd,a,step)
       fx=ffx(1)
       fxd=ffxd(1)
       gx=ggx(1)
       gxd=ggxd(1)
       call rcwfn(yk,fo1,0,0,ffx,ffxd,ggx,ggxd,a,step)
       fy=ffx(1)
       fyd=ffxd(1)
       gy=ggx(1)
       gyd=ggxd(1)
       tgrand=wavek*gibbs*(fx*fyd-fy*fxd)/(twopisq*r12)
c      write(7,'(2x,e14.7,2x,e14.7,2x,"*")') wavek,tgrand/gibbs
c      call flush(7)
       return

1      if(x.eq.0.0) go to 2
       call rcwfn(xk,fo1,0,0,ffx,ffxd,ggx,ggxd,acura,step)
       fx=ffx(1)
       fxd=ffxd(1)
       gx=ggx(1)
       gxd=ggxd(1)
       fo2=wavek**2*gibbs*(fxd**2+
      $        (1.-z/(wavek*wavek*x))*fx*fx)/twopisq
       tgrand=fo2
c      write(7,'(2x,e14.7,2x,e14.7,2x,"*")') wavek,tgrand/gibbs
c      call flush(7)
       return

2      tgrand=(z/twopi)*wavek*gibbs/(exp(z*pi/wavek)-1.)
c      write(7,'(2x,e14.7,2x,e14.7,2x,"*")') wavek,tgrand/gibbs
c      call flush(7)
       return
       end

       subroutine rcwfn(rho,eta,minl,maxl,fc,fcp,gc,gcp,accur,step)
c*********************************************************************
c --- coulomb wavefunctions calculated at r = rho by the             *
c --- continued-fraction method of steed                             *
c --- minl,maxl are actual l-values                                  *
```

```
c --- see barnett feng steed and goldfarb computer physics commun 1974*
c**********************************************************************
      implicit integer (i-n)
      implicit real*8 (a-h,o-z)
      real*8   k,k1,k2,k3,k4,m1,m2,m3,m4
      dimension fc(1),fcp(1),gc(1),gcp(1)

      if(abs(rho).lt.1.e-6) go to 100
      pace = step
      acc = accur
      if(pace.lt.100.0) pace = 100.0
      if(acc.lt.1.0e-15.or.acc.gt.1.0e-6) acc = 1.0e-6
      r    = rho
      ktr = 1
      lmax = maxl
      lmin1= minl + 1
      xll1 = float(minl*lmin1)
      eta2 = eta*eta
      turn = eta + sqrt(eta2 + xll1)
      if(r.lt.turn.and.abs(eta).ge.1.0e-6) ktr = -1
      ktrp = ktr
      go to 2
1     r   = turn
      tf  = f
      tfp = fp
      lmax = minl
      ktrp = 1
2     etar = eta*r
      rho2 =  r*r
      pl  = float(lmax + 1)
      pmx  = pl + 0.5
c --- continued fraction for fp(maxl)/f(maxl)  xl is f  xlprime is fp **
      fp = eta/pl + pl/r
      dk  = etar*2.0
      del = 0.0
      d  = 0.0
      f   = 1.0
      k   = (pl*pl - pl + etar)*(2.0*pl - 1.0)
      if(pl*pl+pl+etar.ne.0.0) go to 3
      r  = r + 1.0e-6
      go to 2
3     h  = (pl*pl + eta2)*(1.0 - pl*pl)*rho2
      k  = k + dk + pl*pl*6.0
      d  = 1.0/(d*h + k)
      del = del*(d*k - 1.0)
      if(pl.lt.pmx) del = -r*(pl*pl + eta2)*(pl + 1.0)*d/pl
      pl  = pl + 1.0
      fp  = fp + del
      if(d.lt.0.0) f = -f
      if(pl.gt.20000.) go to 11
      if(abs(del/fp).ge.acc) go to 3
      fp  = f*fp
      if( lmax.eq.minl) go to 5
      fc (lmax+1) = f
      fcp(lmax+1) = fp
c *** downward recursion to minl for f and fp, arrays gc,gcp are storage
      l  = lmax
      do 4 lp  = lmin1,lmax
      pl = float(l)
      gc (l+1) = eta/pl + pl/r
      gcp(l+1) = sqrt(eta2 + pl*pl)/pl
      fc (l)   = (gc(l+1)*fc(l+1) + fcp(l+1))/gcp(l+1)
      fcp(l)   = gc(l+1)*fc(l)   - gcp(l+1)*fc(l+1)
4     l  = l - 1
      f  = fc (lmin1)
      fp = fcp(lmin1)
5     if(ktrp.eq.-1) go to 1
c --- repeat for r = turn if rho lt turn
c --- now obtain p + i.q for minl from continued fraction (32)
c --- real arithmetic to facilitate conversion to ibm using real*8
      p  = 0.0
      q  = r - eta
      pl = 0.0
      ar = -(eta2 + xll1)
      ai =   eta
      br = 2.0*q
      bi = 2.0
      wi = 2.0*eta
      dr =  br/(br*br + bi*bi)
      di =  -bi/(br*br + bi*bi)
      dp = -(ar*di + ai*dr)
      dq =  (ar*dr - ai*di)
6     p  = p + dp
      q  = q + dq
      pl = pl + 2.0
      ar = ar + pl
      ai = ai + wi
      bi = bi + 2.0
      d = ar*dr - ai*di + br
      di = ai*dr + ar*di + bi
      t = 1.0/(d*d + di*di)
      dr = t*d
      di = -t*di
      h  = br*dr - bi*di - 1.0
      k  = bi*dr + br*di
      t  = dp*h  - dq*k
      dq = dp*k  + dq*h
      dp = t
      if(pl.gt.46000.) go to 11
      if(abs(dp)+abs(dq).ge.(abs(p)+abs(q))*acc) go to 6
      p  = p/r
      q  = q/r
c --- solve for fp,g,gp and normalise f  at l=minl
      g  = (fp - p*f)/q
      gp = p*g - q*f
      w  = 1.0/sqrt(fp*g - f*gp)
      g  = w*g
      gp = w*gp
      if(ktr.eq.1) go to 8
      f  = tf
      fp = tfp
      lmax = maxl
c --- runge-kutta integration of g(minl) and gp(minl) inwards from turn
c ---               see fox and mayers 1968 pg 202
      if(rho.lt.0.2*turn) pace = 999.0
      r3 = 1.0/3.0
      h  = (rho - turn)/(pace + 1.0)
      h2 = 0.5*h
      i2 = idint(pace + 0.001)
      etah = eta*h
      h2ll = h2*xll1
      s  = (etah + h2ll/r  )/r   - h2
7     rh2= r + h2
      t  = (etah + h2ll/rh2)/rh2 - h2
      k1 = h2*gp
      m1 = s*g
      k2 = h2*(gp + m1)
      m2 = t*(g  + k1)
      k3 = h*(gp + m2)
      m3 = t*(g  + k2)
      m3 =    m3 + m3
      k4 = h2*(gp + m3)
      rh = r + h
      s  = (etah + h2ll/rh )/rh  - h2
      m4 = s*(g + k3)
      g  = g  + (k1 + k2 + k2 + k3 + k4)*r3
      gp = gp + (m1 + m2 + m2 + m3 + m4)*r3
      r  = rh
      i2 = i2 - 1
      if(abs(gp).gt.1.0d300) go to 11
      if(i2.ge.0) go to 7
      w  = 1.0/(fp*g - f*gp)
c --- upward recursion from gc(minl) and gcp(minl),stored values are r,s
c --- renormalise fc,fcp for each l-value
8     gc (lmin1) = g
      gcp(lmin1) = gp
      if(lmax.eq.minl) go to 10
      do  9  l = lmin1,lmax
      t        = gc(l+1)
      gc (l+1) = (gc(l)*gc (l+1) - gcp(l))/gcp(l+1)
      gcp(l+1) =  gc(l)*gcp(l+1) - gc(l+1)*t
      fc (l+1) = w*fc (l+1)
9     fcp(l+1) = w*fcp(l+1)
      fc (lmin1) = fc(lmin1)*w
      fcp(lmin1) = fcp(lmin1)*w
      return
10    fc (lmin1) = w*f
      fcp(lmin1) = w*fp
      return
11    w  = 0.0
      g  = 0.0
      gp = 0.0
      go to 8
100   fc(1)=0.0
      fcp(1)=0.0
      if(eta.lt.50.)fcp(1)=sqrt((tpi*eta)/(exp(tpi*eta)-1.))
      data tpi/6.283185307179/
      return
      end



c**********************************************************************
c Follow some subroutines from Numerical Recipes
c**********************************************************************



      SUBROUTINE QSIMP(FUNC,A,B,S)
c**********************************************************************
c   returns as S the integral of the function FUNC from A to B.
c   The parameters EPS can be set to the desired relative accuracy and
c   JMAX so that 2**(JMAX-1) is the maximum allowed number of steps.
c   Integration is performed by Simpson's rule
c   Numerical Recipes pg. 113
c**********************************************************************
      implicit integer (i-n)
      implicit real*8 (a-h,o-z)
      external func
      PARAMETER (EPS=1.D-6, JMAX=20)
      OST=-1.D30
      OS= -1.D30
      DO 11 J=1,JMAX
        CALL TRAPZD(FUNC,A,B,ST,J)
        S=(4.*ST-OST)/3.
        IF (DABS(S-OS).LT.EPS*DABS(OS)) RETURN
      write(3,'(" j del int=",i5,2x,2(d14.7,2x))') j,s-os,s
      write(*,'(" j del int=",i5,2x,2(d14.7,2x))') j,s-os,s
      call flush(3)
        OS=S
        OST=ST
11    CONTINUE
      print*, 'Qsimp: Too many steps.'
      stop
      END

      SUBROUTINE QROMB(FUNC,A,B,SS)
c**********************************************************************
c   returns as SS the integral of the function FUNC from A to B.
c   Integration is performed by Romberg's method of order 2K where
c   e.g. K=2 is simpson's rule.
c   Numerical Recipes page 114
c**********************************************************************
      implicit integer (i-n)
      implicit real*8 (a-h,o-z)
      external func
      PARAMETER(EPS=1.D-6,JMAX=20,JMAXP=JMAX+1,K=5,KM=4)
c**********************************************************************
c   Eps is the fractional accuracy desired as determined by the
c   extrapolation error estimate. JMAX limits the total number of steps
c   and K is the number of points used in the extrapolation
c**********************************************************************
```

```
      DIMENSION S(JMAXP),H(JMAXP)
c*********************************************************************
c   S and H store the succesive trapezoidal approximations and their
c   relative step sizes
c*********************************************************************
      H(1)=1.DO
      DO 11 J=1,JMAX
        CALL TRAPZD(FUNC,A,B,S(J),J)
        IF (J.GE.K) THEN
          L=J-KM
          CALL POLINT(H(L),S(L),K,0.DO,SS,DSS)
          IF (DABS(DSS).LT.EPS*DABS(SS)) RETURN
        ENDIF
        S(J+1)=S(J)
        H(J+1)=0.25*H(J)
11    CONTINUE
      print*, 'Qromb: Too many steps.'
      stop
      END

      SUBROUTINE TRAPZD(FUNC,A,B,S,N)
c*********************************************************************
c   This routine computes the N'th stage of refinement of an extended
c   trapezoidal rule.
c   FUNC is input as the name of the function to be integrated between
c   limits A and B ,also input. When called with N=1, the routine
c   returns as S the crudest estimate of the integral. Subsequent calls
c   with N=2,3....(in that sequential order) will improve the accuracy
c   of S by adding 2**N-2 additional interior points.
c   S should not be modified between call
c   Numerical Recipes  page 111
c*********************************************************************
      implicit integer (i-n)
      implicit real*8 (a-h,o-z)
      external func
      IF (N.EQ.1) THEN
        S=0.5*(B-A)*(FUNC(A)+FUNC(B))
        IT=1
      ELSE
        IT=2**(N-2)
        TNM=1T
        DEL=(B-A)/TNM
        X=A+0.5*DEL
        SUM=0.DO
        DO 11 J=1,IT
          SUM=SUM+FUNC(X)
          X=X+DEL
11      CONTINUE
        S=0.5*(S+(B-A)*SUM/TNM)
      ENDIF
      RETURN
      END

      SUBROUTINE POLINT(XA,YA,N,X,Y,DY)
c*********************************************************************
c   Given arrays XA and YA, each of length N, and given a value X,
c   this routine returns a value Y, and an error estimate DY.
c   If P(x) is a polynomial of degree N-1 such that P(XA(I))=YA(I),
c   I=1,...,N then the returned value Y=P(X).
c   Change NMAX as desired to be the largest anticipated value of N
c   Numerical Recipes page 82
c*********************************************************************
      implicit integer (i-n)
      implicit real*8 (a-h,o-z)
      PARAMETER (NMAX=30)
      DIMENSION XA(N),YA(N),C(NMAX),D(NMAX)
      NS=1
      DIF=DABS(X-XA(1))
      DO 11 I=1,N
        DIFT=DABS(X-XA(I))
        IF (DIFT.LT.DIF) THEN
          NS=I
          DIF=DIFT
        ENDIF
        C(I)=YA(I)
        D(I)=YA(I)
11    CONTINUE
      Y=YA(NS)
      NS=NS-1
      DO 13 M=1,N-1
        DO 12 I=1,N-M
          HO=XA(I)-X
          HP=XA(I+M)-X
          W=C(I+1)-D(I)
          DEN=HO-HP
          IF(DEN.EQ.0.DO)THEN
             print*, 'Problem in Polint'
             stop
          ENDIF
          DEN=W/DEN
          D(I)=HP*DEN
          C(I)=HO*DEN
12      CONTINUE
        IF (2*NS.LT.N-M)THEN
          DY=C(NS+1)
        ELSE
          DY=D(NS)
          NS=NS-1
        ENDIF
        Y=Y+DY
13    CONTINUE
      RETURN
      END
```

```
*********************************************************************************
*** mc-mix.par ******************************************************************
*********************************************************************************

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


         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 ( MBIN = 100000 ) ! maximum dimension of g(r) histogram
      PARAMETER ( MMIX = 2 )      ! maximum number of components
         PARAMETER ( DELR = 0.01d0 ) ! grid spacing for g(r)


      INTEGER*8     DIM, NMIX, ISP(MNP)
         REAL*8        RX(MDIM,0:N,MNP)
         REAL*8        FTM(MMIX), LS, SIGMA
         REAL*8        SIG(MMIX,MMIX),EPSA(MMIX,MMIX),EPSR,EPS,RCUT
         INTEGER*8     FTN0, NP, NPM(MMIX)
      INTEGER*8     NEXT(MNP), LEXT(MNP), MEXT(N,MNP), MBM

         ! path
         COMMON / BLOCK1 / RX, ISP, DIM, NMIX
         ! pair-potential parameters
         COMMON / BLOCK2 / SIG, EPSR, EPSA, EPS, RCUT
         ! mass, timestep, box edge, # timeslices, # particles
         COMMON / BLOCK3 / FTM, LS, SIGMA, FTN0, NP, NPM
         ! permutations
         COMMON / BLOCK4 / NEXT, LEXT, MEXT, MBM

*********************************************************************************
*** data-mix.in *****************************************************************
*********************************************************************************

0 spatial dimensions (1,2,3,...<= MDIM) nixture dimension (1,2,3,...<= MMIX)
3 2
1 seed of the random sequence                                        RAND
3
2 if bose                                                            (T/F)
T
3 number of particles NPM(I)                  (NP=NPM(1)+NPM(2)+...<= MNP)
10
10
4 the potential                                          SUBROUTINE POT
COULOMB
5 # time slices = 1/temperature/timestep                          (< N)
20
6 masses FTM(1), FTM(2), ...                            (hbar = kb = 1)
1.d0
1836.15d0
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 initialization, 1 restart, 2 restart rdf
0
15 density                                               THERMODYNAMICS
.16d0
16 temperature                                           THERMODYNAMICS
.005d0
17 maximum displacement/box edge
.01d0
18 maximum # of bridge timeslices                          (> 1; <= #5)
20
19 potential cutoff distance (LJ)                        SUBROUTINE POT
1.d0
20 sig(i,j)                                 (LJ 2.566)   SUBROUTINE POT
1.d0
-1.d0
-1.d0
1.d0
21 epsr                                     (LJ INIT)    SUBROUTINE POT
1.d0
22 epsa(i,j)                                             SUBROUTINE POT
0.d0
.5d0
.5d0
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
```

[1] D. M. Ceperley, Path integrals in the theory of condensed Helium, Rev. Mod. Phys. **67**, 279 (1995).
[2] 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).
[3] M. H. Kalos and P. A. Whitlock, *Monte Carlo Methods* (John Wiley & Sons Inc., New York, 1986).
[4] R. Fantoni, Coherent State Path Integral Monte Carlo, Eur. Phys. J. D (2025), under review.
[5] D. M. Ceperley, Fermion Nodes, J. Stat. Phys. **63**, 1237 (1991).