Exercises Lecture VIII
Macrostates and microstates: equilibrium and entropy. Metropolis algorithm in the canonical ensemble: verifying Boltzmann's distribution

1. **MC simulation of a simple N-particles model**
   Consider an ideal gas of N non interacting, distinguishable particles, **confined** in a box (fixed **V**) and **isolated** (fixed **E**), divided into left/right with the possibility for one particle at a time to pass through the separation wall, with equal probability from the left to the right or viceversa.

   A **macrostate** is specified for instance by the number of particles on the left side, say n, that can correspond to different **microstates** depending on the list of the specific particles there. A Monte Carlo approach consists in generating a certain number of movements, randomly, and consider them as representative of all the possible movements. The program `box.f90` is a possible implementation of the algorithm describing the time evolution of the system in terms of macrostates, i.e. –given an initial number of particles on the left, n– the approach to equilibrium and what the equilibrium macrostate is.

   (a) Choose N=4, 10, 20, 40, 80, and n=N initially. Make a plot of n (or, better, of n/N) with respect to time. What is the equilibration time τ_{eq} (=how many MC steps)?

   (b) Modify the program so that at each time step t it calculates the number of particles $< n(t) >$ averaged over different runs (e.g. 5 runs). Make a plot to compare $n(t)$ over the individual runs and averaged $< n(t) >$.

   (c) *(Optional; do it at home!)* Compare the numerical value of $< n(t) >$ with the exact analytic results for a simple case, for instance N=4.

   (d) *(Optional)* Consider only one run. Modify the program to calculate numerically the probability P_n of having *at equilibrium* a macrostate with n particles on the left, by simply counting the number of occurring microstates that correspond to the macrostate n and dividing for the total number of microstates generated in the time evolution. Plot the histogram P_n for N=20, 40, 80 and a "sufficiently" long run. Comment.

   (e) Modify the program to measure the statistical fluctuations at the equilibrium, by calculating the variance $\sigma^2 =< n^2 > - < n >^2$, where the average is done over a time interval *after* reaching the equilibrium.

   (f) Determine $< n >$ and $\sigma/ < n >$ at equilibrium for N=20, 40, 80. Which is the dependence of these quantities on N?

   (g) An alternative method to find the equilibrium macrostate is the calculation of the entropy $\mathcal{S}_n$ of the different possible macrostates, by looking at the one with maximum entropy. An efficient numerical implementation is feasible by evaluating the ratio $\mathcal{R}_n$= *sum of possible coincidences for each microstate/maximum number of possible coincidences for each microstate*, then calculating $\mathcal{S}_n \propto -\log \mathcal{R}_n$. The code `entropy.f90` calculates $\mathcal{R}_n$ and $\mathcal{S}_n$. Use it with N=10. Compare the numerically calculated $\mathcal{S}_n$ with the analytical value.

2. **Verification of the Boltzmann distribution** *(see Lecture VII)*

   We can verify directly that the Metropolis algorithm yields the Boltzmann distribution. We consider **a single classical particle** in one dimension in equilibrium with a heath bath (*canonical ensemble*). We fix therefore the temperature T, which labels a *macrostate.* The energy E can vary according to the particular *microstate* (in this particular case, it is enough to label a microstate, a part from the sign of the velocity).

   (a) Write a code (see e.g. `boltzmann_metropolis.f90`) to determine the form of the probability distribution P(E) that is generated by the Metropolis algorithm. Let for instance *T=1*, the initial velocity *vinitial=0*, the number of Monte Carlo steps *nmcs=1000*, and the maximum variation of the velocity *dvmax=2.* Calculate the mean energy, the mean velocity, and the probability density P(E).

   (b) Consider *ln P(E)* as a function of E. Can you recognize the expected behavior ? (see slides for the analytic derivation of P(E)) You should recognise that the asymptotic behavior is a straight line whose slope is $-1/T$.

   (c) How many *nmcs* do you need to have a reasonable estimate of the mean energy and mean velocity ?

   (d) Verify that your results do not depend from the initial conditions by changing *vinitial.* What does it change? What does it changes by changing instead *dvmax* ?

   (e) Modify the program to simulate an ideal gas of **N particles** in one dimension. *[Hint: modify the subroutine Metropolis inserting a loop over the particles]* Consider for instance N=20, T=100, *nmcs*=200. Assume all particles to have the same initial velocity *vinitial*=10. Determine the value of *dvmax* so that the acceptance ratio is about 50% ? What are the mean energy and mean velocity ?

   (f) Calculate P(E), make a plot and describe its behaviour. Is it similar to the case N=1 ? Comment on that.

   (g) Calculate the mean energy for T=10, 20, 30, 90, 100, and 110, and estimate the heat capacity as the *numerical derivative of the energy with respect to the temperature*, $C = \partial < E > /\partial T$.

   (h) Calculate the mean square energy fluctuation $< \Delta E^2 > = < E^2 > - < E >^2$ for T=10 and T=40. Compare the magnitude of the ratio $C = < \Delta E^2 > /T^2$ numerically estimated from the mean square energy fluctuation with that obtained in (f).

3. **Simulated annealing**
   Simulated annealing is a stochastic method for global energy minimization, considering the system starting from a sufficiently high temperature; at each temperature it goes towards equilibrium according to the Boltzmann factor (see the application of the Metropolis algorithm in the canonical ensemble); then the temperature is slightly reduced and the equilibration procedure is repeated, and so on, until a global equilibrium state is reached at T=0. The method can be efficiently used for function minimization, even if the function is not representing an energy. In program `simulated_annealing.f90` it is implemented for the minimization of $f(x) = (x + 0.2) * x + cos(14.5 * x - 0.3)$. Initial temperature, initial position and scaling factor for the temperature are input quantities. Test the program by choosing different initial parameters and scaling factor for the temperature.

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! box.f90
!
! simulation of the evolution of a physical system towards equilibrium:
! non interacting particles in a box divided into two parts;
! at each time step,  one and only one particle (randomly choosen)
! goes from one side to the other one
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
module moduli_box
  implicit none
  public :: initial, move
  integer, public :: N,tmax
contains

  subroutine initial()
    print*," total number of particles N >"
    read*,N
    tmax = 10*N  ! we choose the evolution time proportional to N

  subroutine move()
    integer :: nl,itime
    real :: r, prob
    nl = N ! we start with all the particles on the left side
    open(unit=2,file="box.out",action="write",status="replace")
    do itime = 1,tmax
       prob = real(nl)/N    ! fraction of particles on the left
       call random_number(r)
       if (r <= prob) then
          nl = nl - 1
       else
          nl = nl + 1
       end if
       write(unit=2,fmt=*)nl
    end do
    close(unit=2)
  end subroutine move
end module moduli_box

program box
  use moduli_box
  ! compare a random number with the fraction of particles on the left, nl/N:
  ! if r.le.nl/N we move one particle from the left to the right;
  ! elsewhere from the right to the left
  call initial()
  call move()
end program box
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! entropy.f90
!
! calculates the entropy for each macrostate
! using the "coincidence method" of Ma
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
module ma
  implicit none
  public :: start
  integer, public :: nl,nr,nexch,N
  integer, dimension(10), public :: mleft=0, mright=0
  integer, dimension(:), public, allocatable :: micro

contains

  subroutine start()
    !     initialize parameters
    integer :: il,ir
    print*, " total number of particles N (<=10)>"
    read*, N
    print*, " number of particles 0<nl<N initially on the left (MACROstate)>"

    read*,nl
    if(nl<=0.or.nl>=N)then
       print*,' not acceptable, wrong nl'
       stop
       end if
    nr = N - nl  !  number of particles on the right
    print*, " number of exchanges >" ! no. of evolution steps of the macrostate
    read*, nexch
    allocate(micro(0:nexch))
    micro(0) = 0
    write(*,fmt=*)'nleft =',nl
    write(*,fmt=*)'nright=',nr
    do il = 1,nl
       !     list left particles
       mleft(il) = il
       !     quantity characterizing the initial macrostate
       micro(0) = micro(0)*2 + 2
    end do
    do ir = 1,nr
       !     list right particles
       mright(ir) = ir + nl
    end do
!    print*,'microstate(0)=',micro(0)
!    write(*,fmt="(a,i2,a,10(1x,i2))")'nleft =',nl,' labels=',mleft
!    write(*,fmt="(a,i2,a,10(1x,i2))")'nright=',nr,' labels=',mright
  end subroutine start

  subroutine exch()
    !     exchange one particle on the left (ileft)
    !     with one particle on the right (iright)
    real, dimension(2) :: r
    integer :: iexch,ileft,jleft,iright,jright
    do  iexch = 1,nexch
       !        choose randomly the labels of the two particles
       call  random_number(r)
       ileft  = int (r(1)*nl + 1)    ! 1 =< ileft  =< nl
       iright = int (r(2)*nr + 1)    ! 1 =< iright =< nr
       jleft  = mleft (ileft)
       jright = mright(iright)
       mleft (ileft)  = jright  ! new particle on the left
       mright(iright) = jleft   ! new particle on the right
       !     characterizing the microstate:
       micro(iexch) = micro(iexch-1) + 2**jright  - 2**jleft
!    print*,'microstate(',iexch,')=',micro(iexch)
!    write(*,fmt="(a,i2,a,10(1x,i2))")'nleft =',nl,' labels=',mleft
!    write(*,fmt="(a,i2,a,10(1x,i2))")'nright=',nr,' labels=',mright
    end do
  end subroutine exch

  subroutine output()
    !     calculate the ratio of coincidences with respect to the total number
    !     of possible pairs, and consequently entropy
    !
    integer :: ncoin, ncomp, iexch, jexch
    real :: rate,S
    ncoin = 0
    ncomp = nexch*(nexch-1)/2
    !     compare microstates: if coincident, count + 1;
    !     upgrade counter
    do  iexch = 1,nexch-1
       do jexch = iexch+1, nexch
          if (micro(iexch) == micro(jexch)) ncoin = ncoin + 1
       end do
    end do
    !     coincidence ratio
    rate = real(ncoin)/real(ncomp)
    if (rate > 0) then
       S = log(1.0/rate)
       print*, " numerically estimated entropy: S=",S
       else
```

```
      print*, " no coincidences! estimated entroty infinite! "
  end subroutine output

end module ma

program entropy
  use ma
  !     N:     total number of particles
  !     nl:    number of left particles (i.e. the MACROstate)
  !     mleft(),mright(): labels of left and right particles
  !     micro: a "global" label for a microstate, here defined through
  !            mleft() : micro=sum_{il=1,nl} 2**(mleft(il))
  !     nexch: total number of exchanges (evolution steps of the macrostate)
  !            microst.)
  call start()
  !     the macrostates evolves (exchanging particles, the microstate changes)
  call exch()
  !     calculate the fraction of coincidence of microstates over all
  !     the possible coincidences with the microstates and the entropy
  call output()
  deallocate(micro)
end program entropy
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! simulated_annealing.f90
! for function minimization; adapted from U. Schmitt, 2003-01-15
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
PROGRAM anneal
IMPLICIT NONE
INTEGER :: istep, nsteps
REAL, PARAMETER :: scale=0.5 ! should be chosen for specific function
REAL :: func, fx, fx_min, fx_new, temp, tfactor, x, x_min, x_new
REAL, DIMENSION(2) :: rand ! random numbers
x = 1.0; fx = func(x); fx_min = fx  ! starting point for search
PRINT *, 'Starting from x = ', x, ', f(x) = ', fx

PRINT *, 'initial (high) temperature (e.g., 10)?'  ! annealing schedule
READ *, temp
PRINT *, 'annealing temperature reduction factor (e.g., 0.9)?'
READ *, tfactor
PRINT *, 'number of steps per block (equilibration, e.g., 1000)?'
READ *, nsteps

Do WHILE (temp > 1E-5) ! anneal cycle
  DO istep = 1, nsteps
    CALL RANDOM_NUMBER(rand) ! 2 random numbers
    x_new  = x + scale*SQRT(temp)*(rand(1) - 0.5) ! stochastic move
    fx_new = func(x_new) ! new object function value
    IF (EXP(-(fx_new - fx)/temp) > rand(2)) THEN ! success, save
      fx = fx_new
      x = x_new
    END IF
      write(1,fmt=*)temp,x,fx
    IF (fx < fx_min) THEN
      fx_min = fx
      x_min = x
      PRINT '(3ES13.5)', temp, x_min, fx_min
    END IF
  END DO
  temp = temp * tfactor ! decrease temperature
END DO

End PROGRAM anneal

REAL FUNCTION func(x) ! Function to minimize
Implicit NONE
REAL :: x
func = (x + 0.2)*x + cos(14.5*x - 0.3)
END FUNCTION
```