Exercise Lecture X

Lattice gas
Simulated annealing

(results for items in red have to be included in the homework report)

1. **Self-diffusion coefficient in a lattice gas model**

Consider a finite square lattice with sites randomly occupied by particles with a given density ρ. The particles can move randomly to *empty* nearest sites (two particles can not occupy the same site). It is an example of a restricted random walk. A meaningful physical quantity is the *self-diffusion coefficient* D of an individual particle. D is the limit $t \to \infty$ of $D(t)$, where $D(t)$ is given by:

$$D(t) = \frac{1}{2dt}\langle \Delta R^2(t) \rangle,$$

with d which is the dimensionality of the system and $\langle \Delta R^2(t) \rangle$ is the net *instantaneous* mean square displacement per particle, averaged over all particles, after t units of time *($\langle ... \rangle$ here indicates the average over particles and not temporal averages).*
The dynamical model can be summarized by the following algorithm:

i) Occupy at random the $L \times L$ sites of a square lattice with N particles subject to the condition that no double occupancy is allowed, with the desired density $\rho = N/L^2 \leq 1$. Tag each particle, that is, distinguish it from the others, and record its initial position in an array.

ii) At each step choose a particle (randomly, or, alternatively, in an ordered way) and one of its nearest neighbor sites at random. If the neighbor site is empty, the particle moves to this site; otherwise it does not. Loop over the particles.

Note 1: The measure of "time" in this context is arbitrary. The usual definition is that during one unit of time or one Monte Carlo step*, each particle* on average *attempts one jump. Time goes on even if the particles do not move, i.e., the tentative move is not accepted.*
Note 2: Consider periodic boundary conditions, but note that reliable results can be obtained only for $\langle \Delta R^2(t) \rangle < (L/2)^2$ (this sets a limit to number of MC steps). Otherwise, they could be affected by the imposed periodicity.

Do a Monte Carlo simulation to determine D and its dependence on the particles concentration ρ.

See for instance the code `latticegas.f90`. Internal units for Monte Carlo time step and displacement should be preferred. For comparison with a realistic situation, such as for instance diffusion in solids, we may consider Monte Carlo time step equal to 1 *ns* and the unit length to 2 Å, properly rescaling the internal quantities at the end of the calculations.

(a) Study $\langle \Delta R^2(t) \rangle$ as a function of time for a fixed value of $\rho = 0.03$ and for a fixed number of particles (e.g., 13 particles in a 20×20 lattice). What do you see increasing time (within the limit mentioned above)? Make a fit and compare your result (the slope) with the expected behavior of a standard random walk.

(b) Repeat for $\rho = 0.2$.

(c) Plot $D(t)$ as a function of time: after a certain equilibration time, it fluctuates. Calculate the amplitude of the fluctuations as a function of t (from the distribution of data over the particles). These fluctuations remain also by increasing t.

(d) In order to estimate D, which is defined as the limit of $D(t)$ for $t \to \infty$, do a temporal average $\langle D(t) \rangle_t$ ($\langle ... \rangle_t$ *here indicates a temporal average, for instance from 0 to t, or some block average over time).* Plot together $D(t)$ and $\langle D(t) \rangle_t$. Change the seed, do another run and compare $\langle D(t) \rangle_t$ with the previous results. An estimate of D can be obtained by averaging $\langle D(t) \rangle_t$ over different runs.

(e) Better statistics for $D(t)$ (and consequently for D) can be obtained by calculating $\langle \Delta R^2(t) \rangle$ as average over many particles (i.e., for a given ρ, considering a lattice with L as large as possible; it is suggested $L \geq 40$). Verify that fluctuations of $D(t)$ (and the deviations of $\langle D(t) \rangle_t$ over more runs from its mean value) are proportional to the inverse square root of the number of particles.

(f) Study the dependence of D on the concentration, using for instance ρ=0.1, 0.2, 0.3, 0.5, and 0.7. You should find that D is a monotonically decreasing function of ρ. Why?

(g) To gain some insight into this dependence, determine the dependence on ρ of the probability that if a particle jumps to a vacancy at time t, it returns to its previous position at time $t + 1$. Is there a qualitative relation between the density dependence of D and this probability?

2. 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. For instance:

(a) *Annealing rate*: Try different annealing factors (0.8, 0.9, 0.95) with different random seeds. How slowly do you have to anneal to settle down in the global minimum over 90% of the time? Just estimate the quantity, but provide some explanation and data to support your answer.

(b) $\sqrt{T}$ *scaling*: It is often convenient to scale by $\sqrt{T}$ the step size used in the Metropolis algorithm. Why? *Hint, look at the acceptance ratios, and think about diffusion and thermal distributions in parabolic wells*

3. **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)* 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.

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
 latticegas.f90 - from Gould-Tobochnick
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

 program latticegas
  implicit none

  logical, allocatable :: lattice(:,:)
  integer, allocatable :: x(:),y(:)
  double precision, allocatable :: dx(:),dy(:)

  Integer  :: Nsteps,Np,L
  integer  :: istep,isubstep
  integer  :: dir,i,j,nfail,njumps
  integer, dimension(1) :: seed

  integer,  parameter  ::  MAXINT=1000000000  ! Variables  for  counting
  ! allowed    directions
  integer :: free(4),nfree
  integer :: dxtrial(4),dytrial(4)
  integer :: xnew(4),ynew(4)

  Real, dimension(2) :: rnd(2)
  real :: rnd1
  double  precision :: dxsum,dysum,dxsqsum,dysqsum
  double  precision :: t,deltat,drsqave,D,a,help

  ! Set average time  between jumps and jump length Units is  s and cm
  ! although actually this is not needed for the simulation
  deltat=1.d0          ! or 1d-9 in order to consider 1 ns
  a=1.d0                 ! or 2e-8  in order to consider 2  Ang

  print*," Nsteps>"
  read*, Nsteps
  print*," Np>"
  read*, Np
  print*," L>"
  read*, L
  print*," seed>"
  read*, seed

  call random_seed(put=seed)
  print *,'Doing lattice gas walk to',Nsteps,'MC steps, initial seed',seed
  print *,'using',Np,' particles  on a',L,'^2 square lattice'

  if (Np >= L*L) then
```

```
   print *,'Number of particles > number of sites !!!'
   STOP 'Too small lattice'
endif

allocate(lattice(0:L-1,0:L-1))
allocate(x(Np),y(Np))
allocate(dx(Np),dy(Np))

! Mark all positions as empty
do i=0,L-1
   do j=0,L-1
      lattice(i,j) = .false.
   enddo
enddo

! Enumeration of directions: 1=right; 2=left; 3=up; 4=down
dxtrial(1)=+1; dytrial(1)= 0;
dxtrial(2)=-1; dytrial(2)= 0;
dxtrial(3)= 0; dytrial(3)=+1;
dxtrial(4)= 0; dytrial(4)=-1;

Nfail=0; njumps=0;
! Generate particles on lattice
do i=1,Np
   do ! Loop until empty position found
      !  To   be  on  safe  side,   check  that  upper   limit  not  reached
      call random_number(rnd)
      x(i)=int(rnd(1)*L);  if (x(i)>=L) x(i)=L-1;
      y(i)=int(rnd(2)*L);  if (y(i)>=L) y(i)=L-1;
      if (lattice(x(i),y(i))) then
         ! Position already filled, loop to find new trial
         cycle
      else
         lattice(x(i),y(i))=.true.
         !  Success, go  to next particle
         exit
      endif
   enddo
   dx(i)=0.0d0; dy(i)=0.0d0;
enddo

T=0.0;
do istep=0,Nsteps-1 ! Loop over MC steps
   do isubstep=1,Np ! Do all particles on average once every MC step
      ! Pick one particle at random
      call random_number(rnd1)
```

```
   i=int(rnd1*Np)+1; if (i>Np) i=Np;

   ! Find possible directions, store it in free()
   Nfree=0
   do j=1,4
      xnew(j)=x(i)+dxtrial(j);
      if  (xnew(j) >= L) xnew(j)=0; if (xnew(j)<0) xnew(j)=L-1;
      ynew(j)=y(i)+dytrial(j);
      if  (ynew(j) >= L) ynew(j)=0; if (ynew(j)<0) ynew(j)=L-1;
      if (.not. lattice(xnew(j),ynew(j))) then
         ! Success: position free
         nfree=nfree+1
         free(nfree)=j
      endif
   enddo

   ! If no possible directions, get new particle
   If (nfree == 0) then
      nfail=nfail+1
      cycle
   endif
   njumps=njumps+1

   !  Pick  one of  the  possible directions  randomly
   !  Note that  the dir>nfree  check here  really is  needed!
   call random_number(rnd1)
   dir=int(rnd1*nfree)+1; if (dir>nfree) dir=nfree
   j=free(dir)

   ! Now x(i),y(i) is old position and xnew(j),ynew(j) new
   ! Double check that new site really is free
   if (lattice(xnew(j),ynew(j))) then
      print *,'ERROR:    THIS   SHOULD   BE   IMPOSSIBLE'
      print *,i,j,dir,nfree
      print *,free
      print *,x(i),y(i),xnew(j),ynew(j)
      STOP 'ERROR  new  site  bug'
   endif
   !Empty  old  position  and  fill  new
   lattice(x(i),y(i))=.false.
   lattice(xnew(j),ynew(j))=.true.

   X(i)=xnew(j); y(i)=ynew(j);
   dx(i)=dx(i)+dxtrial(j); dy(i)=dy(i)+dytrial(j);
enddo
```

```
      If (mod(istep*Np,1000000) == 0) then
         ! Calculate  and print intermediate results
         ! Get total displacement from dx,dy
         dxsum=0.0d0; dysum=0.0d0;
         dxsqsum=0.0d0; dysqsum=0.0d0;
         do i=1,Np
            dxsum=dxsum+dx(i);   dysum=dysum+dy(i);
            dxsqsum=dxsqsum+dx(i)*dx(i);
            dysqsum=dysqsum+dy(i)*dy(i);
         enddo
         drsqave=(dxsqsum+dysqsum)/(1.0*Np)

         if (t>0.0) then
            !  Get  diffusion coefficient  by  proper scaling
            D=drsqave*a*a/(4*t)
            write(*,fmt='(3(a,1pe10.2))')&
                 'At ',t,' <dR^2>=',drsqave*a*a,' D=',D,' cm^2/s'
         endif

      endif

      t=t+deltat
   enddo

   !  Get   total  displacement   from  dx,dy
   dxsum=0.0d0; dysum=0.0d0;
   dxsqsum=0.0d0; dysqsum=0.0d0;
   do i=1,Np
      dxsum=dxsum+dx(i);   dysum=dysum+dy(i);
      dxsqsum=dxsqsum+dx(i)*dx(i); dysqsum=dysqsum+dy(i)*dy(i);
   enddo
   print *,'dxsum',dxsum,' dysum',dysum
   print *,'dxsqsum',dxsqsum,' dysqsum',dysqsum

   drsqave=(dxsqsum+dysqsum)/(1.0*Np)
   print *,'drsqave',drsqave
   print *,'Number of  failed jumps',nfail,' number of  successes',njumps
   ! Get diffusion  coefficient  by  proper scaling
   D=drsqave*a*a/(4*t)
   write(*,fmt='(a,f6.4,a)')'For Np/L^2=',real(Np)/L**2,' :'
   write(*,fmt='(3(a,1pe10.2))')&
        'at ',t,' <dR^2>=',drsqave*a*a,' D=',D,' cm^2/s'

   deallocate (lattice,x,y,dx,dy)

end  program latticegas
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! 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
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
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
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! entropy.f90
!
! calculates the entropy for each macrostate
! using the "coincidence method" of Ma
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
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
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! simulated_annealing.f90
! for function minimization; adapted from U. Schmitt, 2003-01-15
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
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
```