Exercises Unit VIII: Metropolis - Monte Carlo algorithm for importance sampling integration

1. **Sampling physical quantities with gaussian distribution: direct sampling and Metropolis sampling**
   Consider the quantum harmonic oscillator and its ground state. The exact solution and the expectation values of kinetic, potential and total energy are know analytically, and can be used to compare the numerical results.

   (a) *Direct sampling.* Estimate kinetic energy, potential energy, first moments $\langle x^i \rangle$ of the wavefunction $\psi(x) = Ae^{-x^2/(4\sigma^2)}$ with a sample-mean Monte Carlo calculation of the integral of the expectation values using a sequence of random points *directly* obtained for instance from the `gasdev` subroutine (see previous Lectures). See for instance the code `direct_sampling.f90`. Study the numerical accuracy and the convergence of the previous quantities as a function of the number of sampling points.

   (b) Is the normalization constant A of the wavefunction important for our purposes?

   (c) *Metropolis sampling.* Repeat the sampling using the Metropolis algorithm. See for instance the code `metropolis_sampling.f90`. Evaluate the numerical accuracy and convergence of the more relevant quantities as a function of the number of sampling points.

2. **Correlations**

   (a) Calculate the autocorrelation function $C(j) = \dfrac{\langle x_i x_{i+j} \rangle - \langle x_i \rangle^2}{\langle x_i^2 \rangle - \langle x_i \rangle^2}$ for a sequence or random numbers with a gaussian distribution using the Metropolis method, with different values of δ/σ: 1, 5, 10, 25, 50. Comment the results.

   (b) For a fixed value of σ compare the autocorrelation function for two sequences of random numbers with a gaussian distribution (i) using the Metropolis method and (ii) using some ad-hoc routine, like for instance `gasdev` based on the Box-Muller algorithm. Discuss the results.

3. **Verification of the Boltzmann distribution**

   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 $\langle E \rangle$ (i.e., total energy of the system $\langle E_{tot} \rangle$ divided by the number of particles) and the mean velocity? *[the symbol $\langle\rangle$ indicates temporal(statistical) averages]*

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

   (g) Calculate the total energy E_{tot} 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*, $C = \partial\langle E_{tot}\rangle/\partial T$. *[C is the heat capacity, i.e. referred to the whole system; you may consider, alternatively, the specific heat, referred to a single particle...]*

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

4. **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 which is the equilibrium macrostate.

   (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?

```
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! metropolis.f90
!
! campionamento METROPOLIS di varie quantita' (tra cui l'energia,
! hamiltoniana:         h = -1/2 \nabla^2 + 1/2 x^2),
! confronto stime numeriche con valori analitici attesi,
! su psi^2(x), con  psi(x) = exp(-\beta x^2)
! (beta = 1/4*sigma^2; con beta=.25 => psi^2(x) = costante * gaussiana normale
!  P(x) =  exp(-x**2/(2*sigma**2))/sqrt(2*pi*sigma**2)
!cccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

program metropolis
  implicit none
  integer, parameter :: dp=selected_real_kind(13)
  integer :: i,n
  integer, dimension(1) :: seed
  real(kind=dp):: sigma,beta,etot,ekin,epot,rnd
  real(kind=dp):: x,x1,x2,x3,x4,xp,delta,lnpsi,lnpsip,p,acc
  character(len=13), save :: format1 = "(a7,2x,2f9.5)"

  acc = 0.0_dp
  x1 = 0.0_dp
  x2 = 0.0_dp
  x3 = 0.0_dp
  x4 = 0.0_dp
  ekin = 0.0_dp
  epot = 0.0_dp
  print*, "seed, n, beta, x0, delta"
  read*,   seed(1),n,beta,x,delta
  call random_seed(put=seed)
  sigma=1.0_dp/sqrt(4.0_dp*beta)

  do i=1,n
     ekin = ekin - 0.5_dp * ((2*beta*x)**2 - 2*beta)
     epot = epot + 0.5_dp * x**2
     etot = ekin + epot
     x1 = x1 + x
     x2 = x2 + x**2
     x3 = x3 + x**3
     x4 = x4 + x**4
     !cccccccccccccccccccccccccccccc
     lnpsi = -beta * x**2           !
     call random_number(rnd)        !
     xp = x + delta * (rnd-0.5_dp)  !
     lnpsip = -beta * xp**2         !   metropolis
     p = exp ( 2 * (lnpsip-lnpsi) ) !   algorithm
     call random_number(rnd)        !
     if (p > rnd) then              !
        x = xp                      !
     !cccccccccccccccccccccccccccccc
        acc=acc+1.0_dp
     endif
  enddo

  write(unit=*,fmt=*)"acceptance ratio = ",acc/n
  write(unit=*,fmt=*)"Risultati (simulazione vs.risultato esatto):"
  write(unit=*,fmt=format1)"etot = ",etot/n,1.0_dp/(8.0_dp*sigma**2)&
       +0.5_dp*sigma**2
  write(unit=*,fmt=format1)"ekin = ",ekin/n,1.0_dp/(8.0_dp*sigma**2)
  write(unit=*,fmt=format1)"epot = ",epot/n,0.5_dp*sigma**2
  write(unit=*,fmt=format1)"<x>  = ",x1/n,0.0_dp
  write(unit=*,fmt=format1)"<x^2>= ",x2/n,sigma**2
  write(unit=*,fmt=format1)"<x^3>= ",x3/n,0.0_dp
  write(unit=*,fmt=format1)"<x^4>= ",x4/n,3.0_dp*sigma**4

end program metropolis
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! diretto.f90
!
! campionamento DIRETTO di varie quantita' (tra cui l'energia,
! hamiltoniana:         h = -1/2 \nabla^2 + 1/2 x^2),
! confronto stime numeriche con valori analitici attesi,
! su psi^2(x), con  psi(x) = exp(-\beta x^2)
! (beta = 1/4*sigma^2; con beta=.25 => psi^2(x) = costante * gaussiana normale
!  P(x) =  exp(-x**2/(2*sigma**2))/sqrt(2*pi*sigma**2)
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

Module gaussian
  implicit none
  public :: gasdev

contains

  SUBROUTINE gasdev(x)

    REAL, INTENT(OUT) :: x
    REAL :: rsq, v1, v2
    REAL, SAVE :: g
    LOGICAL, SAVE :: gaus_stored=.false.

    if (gaus_stored) then
       x=g
       gaus_stored=.false.
    else
       do
          call random_number(v1)
          call random_number(v2)
          v1 = 2.0*v1 - 1.0
          v2 = 2.0*v2 - 1.0
          rsq = v1**2 + v2**2
          if (rsq > 0.0 .and. rsq < 1.0) exit
       end do
       rsq  = sqrt(- 2.0*log(rsq)/rsq)
       x = v1*rsq
       g = v2*rsq
       gaus_stored = .true.
    end if

  END SUBROUTINE gasdev

end module gaussian
```

```
program diretto
  use gaussian
  implicit none
  integer, parameter :: dp=selected_real_kind(13)
  integer :: i,n
  integer, dimension(1) :: seed
  real :: rnd
  real(kind=dp):: sigma,beta,etot,ekin,epot
  real(kind=dp):: x,x1,x2,x3,x4
  character(len=13), save :: format1 = "(a7,2x,2f9.5)"

  x1 = 0.0_dp
  x2 = 0.0_dp
  x3 = 0.0_dp
  x4 = 0.0_dp
  ekin = 0.0_dp
  epot = 0.0_dp
  print*, "seed, n, beta ="
  read*,   seed(1),n,beta
  call random_seed(put=seed)
  sigma=1.0_dp/sqrt(4.0_dp*beta)
  do i=1,n
     !cccccccccccccccccccccccccc
     call gasdev(rnd)   !
     x=rnd*sigma        ! campionamento diretto
     !cccccccccccccccccccccccccc!
     ekin = ekin - 0.5_dp * ((2*beta*x)**2 - 2*beta)
     epot = epot + 0.5_dp * x**2
     etot = ekin + epot
     x1 = x1 + x
     x2 = x2 + x**2
     x3 = x3 + x**3
     x4 = x4 + x**4
  end do

  write(unit=*,fmt=*)"Risultati (simulazione verso risultato esatto):"
  write(unit=*,fmt=format1)"etot = ",etot/n,1.0_dp/(8.0_dp*sigma**2)&
       +0.5_dp*sigma**2
  write(unit=*,fmt=format1)"ekin = ",ekin/n,1.0_dp/(8.0_dp*sigma**2)
  write(unit=*,fmt=format1)"epot = ",epot/n,0.5_dp*sigma**2
  write(unit=*,fmt=format1)"<x>  = ",x1/n,0.0_dp
  write(unit=*,fmt=format1)"<x^2>= ",x2/n,sigma**2
  write(unit=*,fmt=format1)"<x^3>= ",x3/n,0.0_dp
  write(unit=*,fmt=format1)"<x^4>= ",x4/n,3.0_dp*sigma**4

end program diretto
```

```fortran
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! boltzmann_metropolis.f90
!
! Metropolis algorithm used as importance-sampling:
! generation of microstates with Boltzmann distribution,
! here for a classical particle in 1D.
! The interesting quantity is the probability P(E)dE for a particle
! to have energy between E and E+dE (here E can label a microstate,
! a part from the sign +/- of the velocity)
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc

module common
  implicit none
  public :: initial, Metropolis, data, probability, averages
  real, public :: E,T,del_E,beta,dvmax,vel,accept
  integer, public, dimension(:), allocatable :: seed
  integer, public :: nbin,nmcs,sizer
  real, public, dimension(:), allocatable :: P
contains

  subroutine initial(nequil,vcum,ecum,e2cum)
    real, intent(out) :: vcum,ecum,e2cum
    integer, intent(out) :: nequil
    print*," number of MC steps >"
    read *, nmcs
    print*," absolute temperature >"
    read *, T
    print*," initial velocity >"
    read *, vel
    print*," maximum variation of the velocity (hint: 4*sqrt(T)=",4*sqrt(T),") >"
    read *, dvmax
    call random_seed(sizer)
    allocate(seed(sizer))
    print *,'Here the seed has ',sizer,' components; insert them (or print "/") >'
    read *, seed
    call random_seed(put=seed)
    beta   = 1/T
    nequil = 0.1 * nmcs   ! WARNING : VERIFY this choice !
    E      = 0.5 * vel * vel
    del_E  = T/20         ! a reasonable width of the bin for the histogram of P(E)
    nbin   = int(4*T / del_E)  ! max. number of bins
    print *,"# T       :",T
    print *,"# <E0>    :",E
    print *,"# <v0>    :",vel
    print *,"# dvmax   :",dvmax
    print *,"# nMCsteps:",nmcs
    print *,"# deltaE  :",del_E
    print *,"# nbin    :",nbin
    open(unit=9,file="boltzmann.dat",status="replace",action="write")
    write(unit=9,fmt=*)"# T       :",T
    write(unit=9,fmt=*)"# <E0>    :",E
    write(unit=9,fmt=*)"# <v0>    :",vel
    write(unit=9,fmt=*)"# dvmax   :",dvmax
    write(unit=9,fmt=*)"# nMCsteps:",nmcs
    write(unit=9,fmt=*)"# deltaE  :",del_E
    write(unit=9,fmt=*)"# nbin    :",nbin
    allocate (P(0:nbin))
    ecum  = 0.0
    e2cum = 0.0
    vcum  = 0.0
    P     = 0.0
    accept= 0.0
  end subroutine initial

  subroutine Metropolis()
    real :: dv,vtrial,de,rnd
    call random_number(rnd)
    dv = (2*rnd - 1) * dvmax              ! trial variation for v
    vtrial = vel + dv                     ! trial velocity v
    de = 0.5 * (vtrial*vtrial - vel*vel)  ! corresponding variation of E
    call random_number(rnd)
    if (de >= 0.0) then
       if ( exp(-beta*de) < rnd ) return ! trial step not accepted
    end if
    vel = vtrial
    accept = accept + 1
    E = E + de
  end subroutine Metropolis

  subroutine data(vcum,ecum,e2cum)
    real, intent(inout) :: vcum,ecum,e2cum
    Ecum = Ecum + E
    E2cum = E2cum + E*E
    vcum = vcum + vel
    call probability()
  end subroutine data

  subroutine probability()
    integer :: ibin
    ibin = int(E/del_E)
    if ( ibin <= nbin )     P(ibin) = P(ibin) + 1
  end subroutine probability
```

```
  subroutine averages(nequil,vcum,Ecum,E2cum)
    integer, intent(in) :: nequil
    real, intent(in) :: vcum,Ecum,E2cum
    real :: znorm, Eave, E2ave, vave, sigma2
    integer :: ibin
    znorm  = 1.0/nmcs
    accept = accept / (nmcs+nequil) ! acceptance ratio
    Eave   = Ecum * znorm   ! average energy
    E2ave  = E2cum * znorm  !
    vave   = vcum * znorm   ! average velocity
    sigma2 = E2ave - Eave*Eave
    print *,"# <E2>num.:",E2ave
    print *,"# <E> num.:",Eave
    print *,"# <E> th. :",T/2
    print *,"# <v>     :",vave
    print *,"# accept. :",accept
    print *,"# sigma   :",sqrt(sigma2)
    write(unit=9,fmt=*)"# <E2>num:",E2ave
    write(unit=9,fmt=*)"# <E> num.:",Eave
    write(unit=9,fmt=*)"# <E> th. :",T/2
    write(unit=9,fmt=*)"# <v>     :",vave
    write(unit=9,fmt=*)"# accept. :",accept
    write(unit=9,fmt=*)"# sigmaE  :",sqrt(sigma2)
    write(unit=9,fmt=*)"# ibin*del_E, P(E)"
    do ibin = 0,nbin
      write(unit=9,fmt=*) ibin*del_E, P(ibin) * znorm
    end do
    close(unit=9)
  end subroutine averages
end module common

program Boltzmann
  use common
  real :: vcum, ecum, e2cum
  integer :: imcs,nequil
  ! parameters and variable initialization
  call initial(nequil,vcum,ecum,e2cum)
  do  imcs = 1 , nmcs + nequil
     call Metropolis()
     ! data accumulation after each Metropolis step
     if ( imcs > nequil ) call data(vcum,ecum,e2cum)
  end do
  call averages(nequil,vcum,Ecum,E2cum)
  deallocate(P)
end program Boltzmann
```

```
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
! 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
!ccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
module moduli_box
  implicit none
  public :: initial, move
  integer, public :: N,tmax
contains
  subroutine initial()
    integer , dimension(8) :: seed   !!! change according to the seed dimension
    print*," total number of particles N >"
    read*,N
    tmax = 10*N  ! we choose the evolution time proportional to N
    print*," seed (1:8) >"
    read *, seed
    call random_seed(put=seed)
  end subroutine initial

  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 tyhe left to the right;
  ! elsewhere from the right to the left
  call initial()
  call move()
end program box
```