

**Exercises Lecture VII:
Metropolis - Monte Carlo algorithm
gaussian and Boltzmann distributions**

1. **Random numbers with gaussian distribution:**

Metropolis algorithm

Here we use the Metropolis algorithm to generate points with the distribution $P(x) = e^{-x^2/(2\sigma^2)}$. The algorithm is implemented for instance in the code `gauss_metropolis.f90`. We consider $\sigma = 1$, but the suggestion is to write the code for a generic σ .

- (a) Start from $x_0=0$ and choose $\delta=5\sigma$ to be the maximum displacement for each step. Execute runs with $n=100, 1000, 10000, 100000$ points, make an histogram of the points generated and compare it with the gaussian distribution. For which n the agreement is satisfactory?
- (b) Choose n which gives a satisfactory result. For σ fixed, change the step size δ (i.e., change the ratio δ/σ). Determine qualitatively the dependence of the *acceptance ratio* on δ/σ . Make a plot. How to choose δ/σ in order to accept from $\approx 1/3$ to $\approx 1/2$ of trial changes?
- (c) By varying n in a more refined way (e.g. from 100 to 10000 with steps of 100), compare the first moments of the distribution obtained numerically with the exact ones analytically calculated with the Gaussian. In particular, focus on the second moment and make a plot of the difference “exact variance - numerical variance” as a function of n .
- (d) For fixed $\sigma = 1$ and $\delta=5\sigma$, determine qualitatively the *equilibration time* (i.e. the number of steps necessary to *equilibrate* the system); a possible criterion is that the numerical estimate of $\langle x^2 \rangle - \langle x \rangle^2$ is close enough to σ^2 , say within 5%.

2. Sampling physical quantities: 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 known 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 Lecture III). 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 (*since variance and kinetic, potential and total energy depend on the second moment $\langle x^2 \rangle$, you should find the same behavior for all these quantities, a part from a scaling factor*).
- (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. (*see the comment in (a)*)
- (d) Compare the behavior of the absolute error on the total energy with respect to the exact value as a function of the number of sampling points in case of *direct* and of *Metropolis* sampling, making a log-log plot. Comment on the results and explain the possible differences (*consider how the points are generated in the two methods. . .*).

3. Correlations

- (a) Calculate the autocorrelation function $C(j) = \frac{\langle x_i x_{i+j} \rangle - \langle x_i \rangle^2}{\langle x_i^2 \rangle - \langle x_i \rangle^2}$ for a sequence of random numbers distributed according to a gaussian using the Metropolis method, testing different values of δ/σ . Comment the results.
- (b) For a fixed value of σ compare the autocorrelation function for two sequences of numbers generated according to a gaussian (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.

4. 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 heat 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 $v_{initial}=0$, 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 $v_{initial}$. What does it change? What does it change 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 $v_{initial}=10$. 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).


```

!cccccccccccccccccccccccccccccccccccc
    acc=acc+1.0_dp
endif
enddo

write(unit=*,fmt=*)"acceptance ratio = ",acc/n
write(unit=*,fmt=*)"# Results (simulation vs. exact results):"
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_sampling

```



```

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 / del_E
  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

```