Methods of Computational Physics - I (PHY637MJ)

Ayush Shenoy (24021014)

November 22, 2025

Contents

Program 1 - Gaussian Integral Estimation

Aim: To estimate the integral $\int_{-\infty}^{+\infty} dx\, e^{-\alpha x^2}$ by the Trapezoidal method to a given precision (by calculating the truncation error from the analytical value) as well as through adaptive subintervals

We will numerically compute the integral $\int_0^\infty \exp(-\alpha x^2)$. As we necessarily have to set some finite upper limit in numerics, there are two sources of error in the obtained result: one from the truncation of the upper limit and the truncation error of the numerical method itself.

To eliminate the former, we choose the upper limit of $4/\sqrt{2\alpha}$ corresponding to a coverage of 8σ in the full integral. As for the latter, know that for a function $f(x)$ integrated over the interval $[a, b]$ with step-size h using the trapezoidal method, the truncation error is given by

$$E_T \leq \frac{(b-a)}{12} h^2 \times \max|f''(x)|$$

Analytically for the gaussian with as seen in the below figure, the absolute value of the second derivative attains its maximum at $x = 2\alpha$, hence

$$E_T \leq \frac{\sqrt{2\alpha}}{3} h^2$$

If δ is the maximum amount of error we are willing to tolerate, we may choose any h such that

$$h^2 \leq \frac{3\delta}{\sqrt{2\alpha}}$$

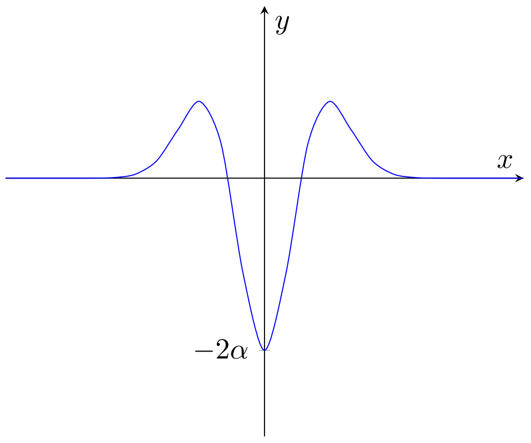


Figure 1: $f''(x) = 4\alpha^2 x^2 e^{-\alpha x^2} - 2\alpha e^{-\alpha x^2}$ for $\alpha = 1$

```
! AYUSH PRAVIN SHENOY
! 24021014
!
! To estimate the gaussian integral by the Trapezoidal method to a given precision
! (by calculating the truncation error from the analytical value) as well as through
! adaptive subintervals

program GINTEG

    use INTEGRATE
    use FORMULA

    implicit none

    real :: ALPHA           ! Gaussian parameters
    real :: THR             ! Error tolerance

    real :: B               ! Integration Upper Limit
    real    ::  TRUVAL      ! Analytical value
    real    ::  INTEG       ! Numerical result
    real    ::  H           ! Subinterval size
    integer ::  N           ! No of subintervals
    real    ::  DELTA

    integer::I

    read(*,*) ALPHA
    read(*,*) THR

    B = 4.0/sqrt(2*ALPHA)           ! 4 Sigma
    TRUVAL = 0.5*sqrt(PI/ALPHA)     ! Half-Integral

    write(*,*) "Alpha               : ", ALPHA
    write(*,*) "Error Threshold (1E): ", -THR
    write(*,*)

    ! Trapezoidal with analytical error bound
    DELTA = 10**(-THR)
    H = sqrt(3.0/sqrt(2*ALPHA))*sqrt(DELTA)
    N = B/H

    write(*,*) "DELTA :", DELTA
    write(*,*) "    H :", H
    write(*,*) "    N :", N
    write(*,*)

    call TRAPEZOID(F,0.0,B,N,INTEG)

    write(*,*) "# ANALYTICAL ERROR BOUND"
    call PRINT_HEADER
    write(*,2) N, INTEG, abs(INTEG-TRUVAL), 100.0*(INTEG-TRUVAL)/TRUVAL
    call PRINT_RESULT(INTEG,TRUVAL)
```

```
! Trapezoidal with adaptive subintervals
write(*,*) "# TRAPEZOIDAL METHOD"
call PRINT_HEADER

N=1
do
    call TRAPEZOID(F,0.0,B,N,INTEG)
    write(*,2) N, INTEG, abs(INTEG-TRUVAL), 100.0*(INTEG-TRUVAL)/TRUVAL
    if (abs(2*(INTEG-TRUVAL)) <= 1E-3) exit
    N = N*2
end do
call PRINT_RESULT(INTEG,TRUVAL)

! Simpson with adaptive subintervals
write(*,*) "# SIMPSONS 1/3 RULE"
call PRINT_HEADER

N=1
do
    call SIMPSON(F,0.0,B,N,INTEG)
    write(*,2) N, INTEG, abs(INTEG-TRUVAL), 100.0*(INTEG-TRUVAL)/TRUVAL
    if (abs(2*(INTEG-TRUVAL)) <= 1E-3) exit
    N = N*2
end do
call PRINT_RESULT(INTEG,TRUVAL)
!write(*,*) H, 2*INTEG, TRUVAL, abs(2*INTEG - TRUVAL)

2 format (2X,I15,4X,3(F15.10,4X))    ! Values

contains

    real function F(X)

        real, intent(in) :: X

        F = exp(-ALPHA*(X**2))
    end function F

    subroutine PRINT_HEADER()
        write(*,1) "#", "N_INTERVALS", "INTEGRAL", "ABS ERROR", "% ERROR"
        write(*,1) "#", ("---------------", I=1,4)

        1 format (A1,1X,4(A15,4X))    ! Table header
    end subroutine

    subroutine PRINT_RESULT(INTEG,TRUVAL)

        real, intent(in) :: INTEG
        real, intent(in) :: TRUVAL

        write(*,*)
        write(*,*) "Converged value  :", 2*INTEG
        write(*,*) "Analytical value :", 2*TRUVAL
```

```
            write(*,*)

        end subroutine

end program GINTEG
```

Flowchart:

Input:

```
1.0
3
```

Output:

```
 Alpha               :    1.00000000
 Error Threshold (1E):   -3.00000000

 DELTA :   1.00000005E-03
     H :   4.60577980E-02
     N :          61

 # ANALYTICAL ERROR BOUND
#     N_INTERVALS           INTEGRAL         ABS ERROR           % ERROR
# ---------------    ---------------    ---------------    ---------------
              61       0.8861705065       0.0000564456      -0.0063692038

 Converged value  :   1.77234101
 Analytical value :   1.77245390

 # TRAPEZOIDAL METHOD
#     N_INTERVALS           INTEGRAL         ABS ERROR           % ERROR
# ---------------    ---------------    ---------------    ---------------
               1       1.4146879911       0.5284610391      59.6304397583
               2       0.8987370133       0.0125100613       1.4116091728
               4       0.8861057162       0.0001212358      -0.0136799999

 Converged value  :   1.77221143
 Analytical value :   1.77245390

 # SIMPSONS 1/3 RULE
#     N_INTERVALS           INTEGRAL         ABS ERROR           % ERROR
# ---------------    ---------------    ---------------    ---------------
               1       0.9431253076       0.0568983555       6.4202919006
               2       0.7267533541       0.1594735980     -17.9946689606
               4       0.8818953037       0.0043316483      -0.4887741506
               8       0.8861675858       0.0000593662      -0.0066987611

 Converged value  :   1.77233517
 Analytical value :   1.77245390
```

Program 2 - Pi Estimation by Acceptance Rejection Method

Aim: To estimate the value of π to within a given precision using the acceptance-rejection method along with ensemble-averaging

In this method, a variation of which was proposed by Laplace in 1812, we actually estimate the volume of a circle $\mathcal{C}$ inscribed within the square region $\mathcal{S} = [-1, +1] \times [-1, +1] \subset \mathbb{R}^2$

We sample two uniformly distributed numbers x_i, y_i, with the distribution scaled such that $x_i, y_i \in [-1, +1] \subset \mathbb{R}$. As the two samples are independent, we may regard a series of N such pairs (x_i, y_i) as uniformly distributed points in $\mathcal{S}$

Given that the distribution of points is uniform, we have for a given sampled point p:

$$\begin{aligned}\mathbf{Prob}(p \in \mathcal{C}) &= \frac{\mathbf{Area}(\mathcal{C})}{\mathbf{Area}(\mathcal{S})} \\ &= \frac{\pi}{4}\end{aligned}$$

The probability in the above expression is estimated by the fraction of points out of N that lie inside the circle. Thus we have

$$\pi = 4 \times \frac{\text{no. of points inside C}}{N}$$

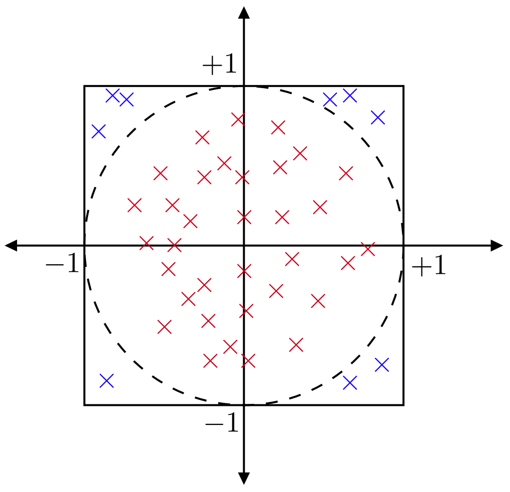


Figure 2: Diagrammatic representation of the acceptance-rejection method.

Flowchart:

Program:

```
! AYUSH PRAVIN SHENOY
! 24021014
!
! To estimate the value of  to within a given precision using the acceptance-rejection
! method along with ensemble-averaging

program PIMONTEC

    use iso_fortran_env, only: IK => int32, RK => real64

    use MONTECARLO

    implicit none

    integer ::  N        ! ensemble size
    real    ::  THR      ! ERR < 1.0E-(THR)
    integer ::  M        ! Number of throws

    integer ::  J
    real    ::  GUESS
    real    ::  ENSAV

    double precision,parameter::PI=4*atan(1.d0)

    read(*,*) THR
    read(*,*) N
    read(*,*) M

    write(*,*) "Error Threshold (1E): ", -THR
    write(*,*)
    write(*,1) "N_THROWS", "GUESS", "ABS ERROR", "PERCENT ERROR"
    write(*,1) ("----------------", J=1,4)

    THR = 10**(-THR)

    do

        ! Populate ensemble
        ENSAV=0
        do J=1,N
            call ACCREJ_SPHERE(1.0,2,M,GUESS)
            ENSAV = ENSAV + GUESS
        end do
        ENSAV = ENSAV/N

        ! Output results
        write(*,2) M, ENSAV, ENSAV-PI, 100*(ENSAV-PI)/PI

        ! Check for convergence
        if (abs(ENSAV - PI) .lt. THR) exit
        M = 10*M

    end do

    write(*,*)
    write(*,*) "Converged value of PI is: ", ENSAV

1 format (A15,4X,3(A15,4X))
2 format (I15,4X,3(F15.10,4X))

end program PIMONTEC
```

Input:

```
4
100
10
```

Output:

```
 Error Threshold (1E):   -4.00000000

       N_THROWS              GUESS          ABS ERROR      PERCENT ERROR
---------------    ---------------    ---------------    ---------------
             10       3.1279997826      -0.0135928710      -0.4326745230
            100       3.1311995983      -0.0103930553      -0.3308212242
           1000       3.1408393383      -0.0007533153      -0.0239787703
          10000       3.1411597729      -0.0004328807      -0.0137790212
         100000       3.1412582397      -0.0003344138      -0.0106447233
        1000000       3.1415793896      -0.0000132640      -0.0004222068

 Converged value of PI is:    3.14157939
```

Program 4 - Gaussian Integral by Monte Carlo Integration

Aim: To estimate the value of a gaussian integral to within a given precision using uniform-sample Monte Carlo integration along with ensemble-averaging

Consider an integral that is perhaps analytically intractable:

$$I = \int_a^b dx\ f(x)$$

The general Monte-Carlo integration problem is to write I in the form

$$I = \int_a^b dx\ g(x)h(x)dx$$

where is a probablity distribution $g(x)$ we have an efficient method to sample from. We can then note that the integral we wish to solve is simply the expectation value of the function $h(X)$ where X is distributed as $g(x)$. This expectation value can now be estimated by drawing say N samples ξ_i from $g(x)$, thus yielding

$$I = \langle h \rangle \approx \frac{1}{N}\sum_{i=1}^{N} h(\xi_i)$$

In our case we wish to solve the integral

$$I_\alpha = \int_0^t dx\ e^{-\alpha x^2}$$

Applying the above procedure, we can insert a uniform distribution (scaled from 0 to t), $U(x) = 1/t$ and obtain

$$I = \frac{t}{N}\sum_{i=1}^{N} e^{-\alpha \xi_i^2}$$

To compute the error in estimation, we note that the well-tabulated error-function is given by

$$\mathrm{erf}(t) = \frac{2}{\sqrt{\pi}}\int_0^t dx\ e^{-\alpha x^2}$$

$$\implies I_\alpha = \sqrt{\frac{\pi}{4\alpha}}\mathrm{erf}(t)$$

Flowchart:

```
! AYUSH PRAVIN SHENOY
! 24021014
!
! To estimate the value of a gaussian integral to within a given precision using
! uniform-sample Monte Carlo integration along with ensemble-averaging


program MCINTEG

    use FORMULA

    implicit none

    real    ::  A           ! Gaussian parameter
    real    ::  T           ! Interval Endpoint
    integer ::  N           ! Number of samples
    real    ::  THR         ! Convergence threshold
    integer ::  M           ! Ensemble size

    real::INTEG
    integer::I
    real::TRUVAL
    real,allocatable::R(:)


    read(*,*) A
    read(*,*) T
    read(*,*) N
    read(*,*) THR
    read(*,*) M

    write(*,*) "Error Threshold (1E): ", -THR
    write(*,*)
    write(*,1) "ENSEMBLE SIZE", "INTEGRAL", "ABS ERROR", "PERCENT ERROR"
    write(*,1) ("----------------", I=1,4)

    allocate(R(N))

    THR = 10**(-THR)
    TRUVAL = 0.5*sqrt(PI/A)*erf(T)

    do
        INTEG = 0
        do I=1,M
            call RANDOM_NUMBER(R)
            R = R*T                             ! Scale dist to T
            R = GAUSSIAN(R,A)                   ! Evaluate function
            INTEG = INTEG + sum(R)*T/N          ! Calculate integral
        end do
        INTEG = INTEG/M

        write(*,2) M, INTEG, INTEG-TRUVAL, 100.0*(INTEG - TRUVAL)/TRUVAL
        ! Check convergence
        if (abs(INTEG-TRUVAL) .lt. THR) exit

        ! Adjust ensemble size
        M = 2*M

    end do

    write(*,*)
    write(*,*) "The converged value is: ", INTEG
    write(*,*) "The tabulated value is: ", TRUVAL

    1 format (A15,4X,3(A15,4X))
    2 format (I15,4X,3(F15.10,4X))

end program MCINTEG
```

Input:

```
2.0
5
10000
4
10
```

Output:

```
 Alpha              :    2.00000000
 Error Threshold (1E):   -4.00000000

  ENSEMBLE SIZE          INTEGRAL          ABS ERROR      PERCENT ERROR
---------------    ---------------    ---------------    ---------------
             10       0.6262360811      -0.0004209876      -0.0671799034
             20       0.6258510351      -0.0008060336      -0.1286243498
             40       0.6226068735      -0.0040501952      -0.6463176608
             80       0.6238328218      -0.0028242469      -0.4506846070
            160       0.6251795888      -0.0014774799      -0.2357716858
            320       0.6275514364       0.0008943677       0.1427204311
            640       0.6267522573       0.0000951886       0.0151899057

 The converged value is:   0.626752257
 The tabulated value is:   0.626657069
```

Program 5 - Box-Muller Transformation

Aim: To sample two uniformly distributed random variables and calculate their Box-Muller transform, verifying that the result is normally distributed.

The Box-Muller transform of two uniformly distributed random variables u and v is given by

$$\begin{aligned} x(u,v) &= \cos(2\pi v)\sqrt{-2\ln u} \\ y(u,v) &= \sin(2\pi v)\sqrt{-2\ln u} \end{aligned}$$

Flowchart:

```
! AYUSH PRAVIN SHENOY
! 24021014
!
! To sample two uniformly distributed random variables and calculate their Box-Muller
! transform, verifying that the result is normally distributed

program BOXMULLER

    use FORMULA

    implicit none

    integer ::  N               ! Number of samples

    integer          :: I
    real             :: M(2,2)
    real,allocatable :: O(:,:)
    real,allocatable :: R(:,:)

    read (*,*) N

    allocate(O(N,2))
    allocate(R(N,2))
    call RANDOM_NUMBER(O)

    R(:,1) = sqrt(-2*log(O(:,1)))*cos(2*PI*O(:,2))
    R(:,2) = sqrt(-2*log(O(:,1)))*sin(2*PI*O(:,2))

    do I=1,N
        write(*,1) O(I,:), R(I,:)
    end do

    M(1,1) = sum(R(:,1))/N
    M(1,2) = sum(R(:,1)**2)/N - M(1,1)**2

    M(2,1) = sum(R(:,2))/N
    M(2,2) = sum(R(:,2)**2)/N - M(2,1)**2

    write(*,2) "#"
    write(*,2) "#", "X", "Y"
    write(*,2) "#", ("--------------", I =1,2)
    write(*,3) "# Mean : ", M(1,1), M(2,1)
    write(*,3) "# Sdev : ", M(2,1), M(2,2)

    1 format(4(F15.10,4X))
    2 format(A1,8X,A15,4X,A15)
    3 format(A9,F15.10,4X,F15.10)

end program BOXMULLER
```

Output:

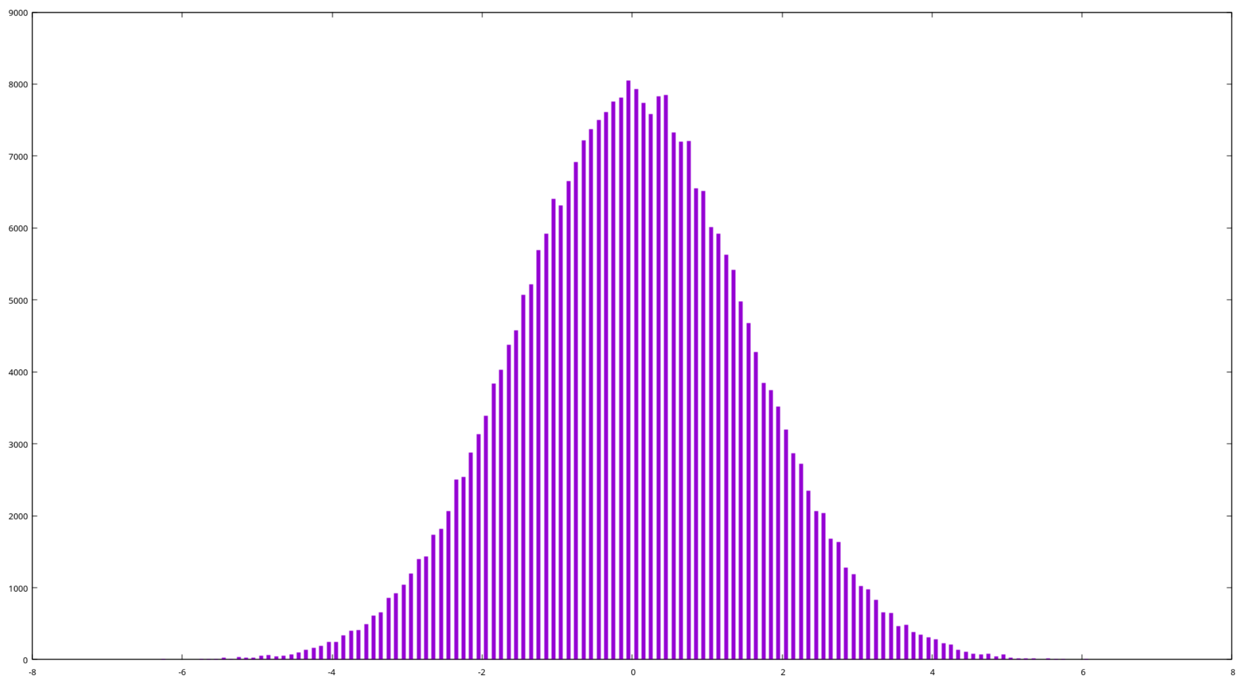


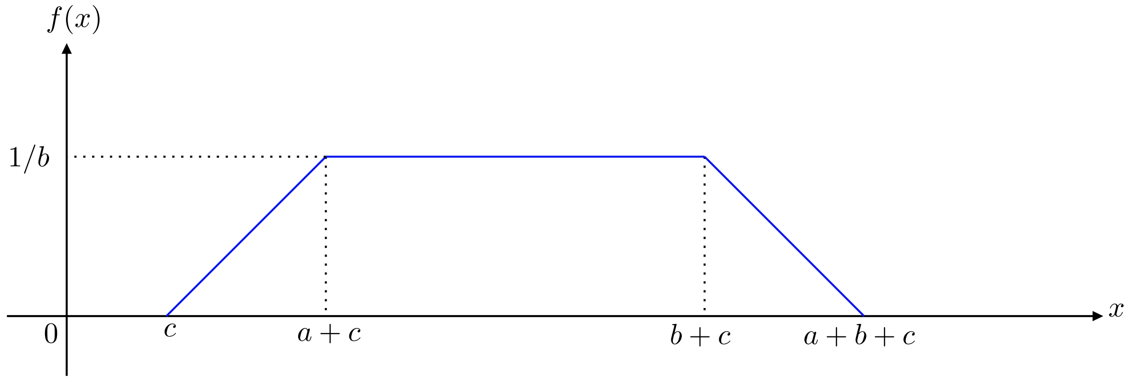
Figure 3: Histogram of $n = 10000$ Box-Muller transformed variables

Program 6 - Trapezoidal distribution

Aim: To sample the given trapezium-shaped distribution by implementing the appropriate transformation of a uniformly distributed random variable.

The given distribution is

$$t(x) = \begin{cases} (x-c)/ab & c \le x \le a+c \\ 1/b & a+c \le x \le b+c \\ (a+b+c-x)/ab & b+c \le x \le a+b+c \end{cases}$$



By integration, the CDF is found to be

$$F_t(x) = \begin{cases} \dfrac{x^2}{2ab} - \dfrac{c}{ab}x + \dfrac{c^2}{2ab} & c \le x \le a+c \\ \dfrac{a}{2b} + \dfrac{x}{b} - \dfrac{a+c}{b} & a+c \le x \le b+c \\ 1 - \dfrac{(a+b+c-x)^2}{2ab} & b+c \le x \le a+b+c \end{cases}$$

Given that we can efficiently sample a uniformly distributed random variable Y, our method is to find $F_t^{-1}(Y)$ which we know is distributed as $t(y)$. In our case we calculate analytically:

$$F_t^{-1}(Y) = \begin{cases} c + \sqrt{2abY} & 0 \le x \le \dfrac{a}{2b} \\ bY + \dfrac{a}{2} + c & \dfrac{a}{2b} \le x \le 1 - \dfrac{a}{2b} \\ a+b+c - \sqrt{2ab(1-Y)} & 1 - \dfrac{a}{2b} \le x \le 1 \end{cases}$$

The above piecewise function of a uniformly distributed random variable is straightforwardly implemented in Fortran. As shown in the below diagram, it is distributed as desired.

Flowchart:

```fortran
program TRAP_DIST

    implicit none


    integer :: N
    real    :: A
    real    :: B
    real    :: C

    real,allocatable    ::  U(:)
    real,allocatable    ::  R(:)
    real                ::  D
    integer             ::  I

    ! Read parameters
    read (*,*) N
    read (*,*) A
    read (*,*) B
    read (*,*) C


    allocate(U(N))
    allocate(R(N))
    call RANDOM_NUMBER(U)

    D = A/(2*B)

    ! Calculate inverse CDF
    do I = 1,N

        if ( U(I) >= 0 .and. U(I) <= D ) then
            R(I) = C + sqrt(2.0*A*B*U(I))
        elseif ( U(I) > D .and. U(I) < (1- D)) then
            R(I) = B*U(I) + 0.5*A + C
        elseif ( U(I) > (1-D) .and. U(I) <= 1.0 ) then
            R(I) = A+B+C - sqrt(2*A*B*(1-U(I)))
        endif

        write(*,1) U(I), R(I)

    end do

    1 format(2(F15.10,4X))

end program TRAP_DIST
```

Output:

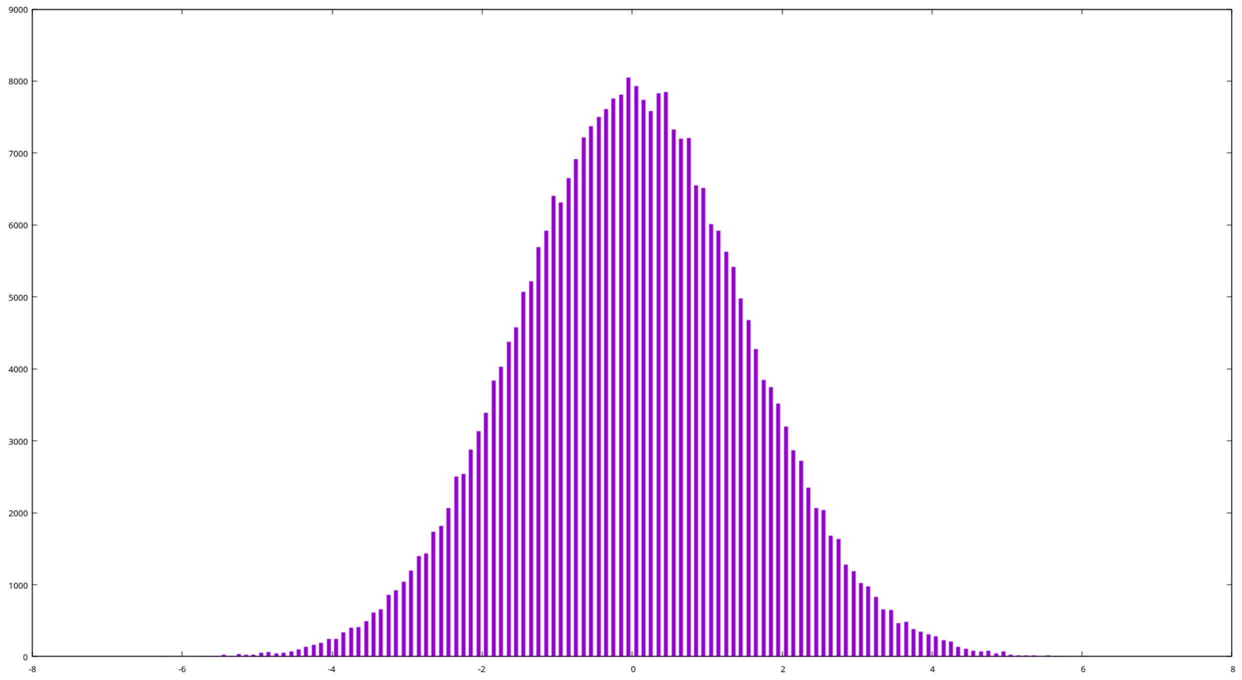


Figure 4: Histogram of $n = 10000$ Box-Muller transformed variables

Program 7 - Ising Model

Aim: To simulate the zero-field Ising model using the Metropolis-Hasting algorithm and calculate the energy, magnetization, magnetic susceptibility and specific heat as a function to temperature and lattice size.

The Ising model is a mathematical model for a ferromagnetic material, defined by placing spins that can either point up or down on a lattice of finite size, say L. To overcome the necessarily finite size in simulation, periodic boundary conditions are imposed. The system is considered to be placed in equilibrium with a bath of temperature T.

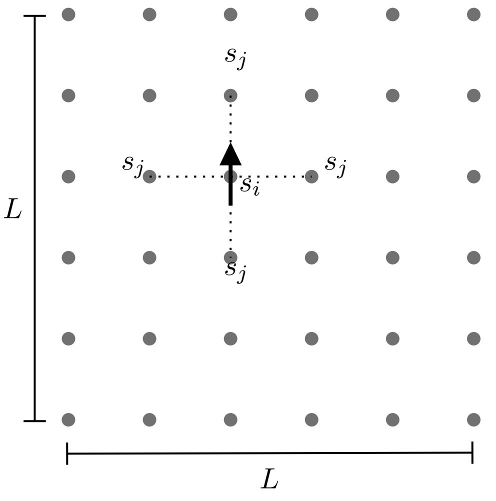


Figure 5: A spin s_i on the lattice and its nearest neighbours s_j

The Hamiltonian is defined to be

$$H = -J \sum_{\langle i,j \rangle} s_i s_j - G \sum_i s_i$$

where the sum $\langle ij \rangle$ is over nearest neighbours as shown in the above diagram. J represents the coupling strength between individual spins while G represents the interaction of the spins with a honmogenous external magnetic field. Observables of interest are the total energy E, total magnetization M, as well as the magnetic susceptibility χ and heat capacity C_V:

$$\chi = \langle M^2 \rangle - \langle M \rangle^2$$
$$C_V = \langle E^2 \rangle - \langle E \rangle^2$$

As the system is liable to exhibit thermal fluctuations, we must calculate ensemble averages of the above observables. However owing to the large (2^{L^2}) number of configurations, this is not viable. Hence we resort to Monte Carlo techniques, sampling possible configurations in order to calculate the averages.

As the system is in equilibrium with a heat-bath we expect the configurations to follow Boltzman statistics. We use the Metropolis method to generate a finite of configurations that are representative of the entire phase space, thus yielding the correct ensemble averages. This is done using Markov chains.

The master equation for a Markov process is

$$\rho(X,t+1)-\rho(X,t)=-\sum_{X'}T(X\to X')\rho(X,t)+\sum_{X'}T(X'\to X)\rho(X',t)$$

We are interested in the stationary distribution $\rho(X)$ for which the left side vanishes. One possible solution is the detailed balance condition where in the long run the rates of both opposing processes become equal:

$$\frac{T(X\to X')}{T(X'\to X)}=\frac{\rho(X,t)}{\rho(X',t)}$$

We the write the transition probabilities in the form

$$T(X\to X')=A_{XX'}\omega_{XX'}$$

where $0\leq\omega_{XX'}\leq 1$, $\sum_X\sum_{X'}\omega_{XX'}=1$, and $\omega_{XX'}=\omega_{X'X}$
We then have

$$\frac{A_{XX'}}{A_{X'X}}=\frac{\rho(X')}{\rho(X)}$$

Our targetted stationary distribution is the Boltzmann distribution, so

$$\begin{aligned}\frac{\rho(X')}{\rho(X)}&=e^{-\beta(H[X']-H[X])}=e^{-\beta\Delta H}\\ \implies\frac{A_{XX'}}{A_{X'X}}&=e^{-\beta\Delta H}\end{aligned}$$

Flowchart:

```
subroutine PRINT_LATTICE(LATT,U)

    integer, intent(in) :: U            ! Output unit (6 for stdout)
    integer, intent(in) :: LATT(:,:)    ! Lattice to print
    integer             ::  I
    integer             ::  J
    integer             ::  L

    L = size(LATT(1,:))

    do I = 1,L
        write(U,*) (LATT(I,J), J=1,L)
        write(U,*)
    end do
    !write(U,*) ("------",I=1,2*L)
    write(U,*)

end subroutine

subroutine PRINT_OBSERVABLES(TIME,E,E_SQ,M,M_SQ,U)
        ! Write observables to output file

        integer, intent(in) :: TIME
        real,    intent(in) :: E
        real,    intent(in) :: E_SQ
        real,    intent(in) :: M
        real,    intent(in) :: M_SQ
        integer, intent(in) :: U

        105 format (2X,I8,4X,4(F20.10,4X))    ! Observables
        write(U,105) TIME, E ,E_SQ, M, M_SQ

end subroutine

integer function WRAP(X,L)
    ! Wrap co-ordinate around if needed

    integer, intent(in) :: X
    integer, intent(in) :: L

    if (X .eq. 0) then
        WRAP = X + L
    elseif (X .eq. L+1) then
        WRAP = X - L
    else
        WRAP = X
    end if

end function

program ISING_MODEL
```

```
implicit none

! Interfaces and other formalities
interface
subroutine PRINT_LATTICE(LAT, U)
    integer, dimension(:,:), intent(in) :: LAT
    integer, intent(in)                 :: U
end subroutine
end interface
integer,external :: WRAP

! Model Parameters
! -----------------------------------------------------------------
real    :: T_START  ! Starting Temperature          (units of kB)
real    :: T_STOP   ! Ending Temperature            (units of kB)
real    :: T_STEP   ! Temperature step              (units of kB)
real    :: J        ! Spin-Spin coupling strength   (units of kB)
real    :: G        ! Spin-Field coupling strength  (units of J)
integer :: L        ! Lattice size

! Numerical Parameters
! -----------------------------------------------------------------------
real    :: CONV_THR = 1E-3         ! Equilibriation Threshold
integer :: EQ_SAMPLES = 100000     ! Number of post-equilibrium samples


! Observables
! ---------------------------------------------
real :: M    = 0    ! Net magnetization
real :: M_SQ = 0    ! Net magnetization squared
real :: E    = 0    ! Total energy
real :: E_SQ = 0    ! Total energy sq

! Observable Averages
! ---------------------------------------------
real :: MEAN_M    = 0
real :: MEAN_M_SQ = 0
real :: MEAN_E    = 0
real :: MEAN_E_SQ = 0

! Response functions
! -------------------------------------------
real :: CHI         ! Magnetic susceptibility
real :: C_V         ! Specific heat

! Other Variables
! ------------------------------------------------------------------------------------

! IO and Dummy
character (len=4) :: DUMMY
integer           :: I,P,Q,S
logical           :: VERBOSE
```

```
! Simulation
integer, allocatable    :: LATTICE(:,:)         ! Current state
real                    :: T                    ! Current temperature
real                    :: E_DIFF               ! Difference in energy after M-H step
real ,dimension(2)      :: R_SF                 ! Spin-flip roll
real                    :: R_CA                 ! Configuration acceptance roll
integer                 :: X_F                  ! Spin-flip X-coord
integer                 :: Y_F                  ! Spin-flip Y-coord
integer                 :: TIME                 ! Timestep counter


! Model Initialization
! ---------------------------------------------

! Read input file from standard input
read(*,*) T_START
read(*,*) T_STOP
read(*,*) T_STEP
read(*,*) J
read(*,*) G
read(*,*) L

! Open output files
open(unit=1,file="./ising.out")
open(unit=2,file="./observables.out")
open(unit=3,file="./state.out")

! Write inital inputs
write(1,*) "INPUT PARAMETERS"
write(1,*) ("-", I = 1,16)
write(1,103) "T_STRT = ", T_START
write(1,103) "T_STOP = ", T_STOP
write(1,103) "T_STOP = ", T_STEP
write(1,103) "J = ", J
write(1,103) "G = ", G
write(1,104) "L = ", L
write(1,*)

! Write output headers
write(2,101) "#", "TIME", "E", "E_SQ", "M", "M_SQ"
write(2,101) "#", "--------", ("--------------------", I=1,4)

! Initialize system
! ------------------------------------------------
allocate(LATTICE(L,L))

LATTICE = +1

write(1,*) "INITIAL CONFIGURATION"
write(1,*) "---------------------"
write(*,*)
call PRINT_LATTICE(LATTICE,1)
write(*,*)
```

```
! Calculate Hamiltonian
E = - G*sum(LATTICE)     ! Spin-Field Interaction Energy
do P = 1,L                ! Spin-Spin  Interaction Energy
    do Q = 1,L
        E = E - J*LATTICE(P,Q)*(LATTICE(P          , WRAP(Q+1,L))&  ! Right
                                +LATTICE(WRAP(P+1,L), Q         ))   ! Down
    end do
end do

E_SQ = E**2
M = sum(LATTICE)
M_SQ = M**2

!call PRINT_OBSERVABLES(0,E,E_SQ,M,M_SQ,2)

write(1,103) "Initial E   =", E
write(1,103) "Initial M   =", M


T = T_START
write(*,106) "#", "TEMP", "CHI_S", "C_V_S", "E_S", "M_S"
write(*,106) "#", ("--------------------", I=1,5)

! Run T : [T_START,T_STOP] Experiment
! ----------------------------------------------
do

    MEAN_E = 0
    MEAN_M = 0
    MEAN_E_SQ = 0
    MEAN_M_SQ = 0

    TIME = 0

    ! Timesteps
    ! --------

    do TIME = 1, EQ_SAMPLES
    ! Monte Carlo Cycle
    ! ----------------

        ! Metropolis steps
        do I = 1,SIZE(LATTICE)

            ! Propose configuration

            ! Flip a random spin
            call RANDOM_NUMBER(R_SF)
            R_SF = R_SF*L
            R_SF = int(ceiling(R_SF))
            X_F = R_SF(1)
            Y_F = R_SF(2)
            S = LATTICE(X_F,Y_F)
            LATTICE(X_F,Y_F) = -1*S
            !write(3,*) "TEMP", T, "TIME", TIME
            !write(3,*) I, "PROP", X_F, Y_F, S, -S

            ! Calculate spin-spin energy change (Up,Down Left Right)
            E_DIFF = +2*J*S*(LATTICE(WRAP(X_F-1,L),Y_F) + LATTICE(WRAP(X_F+1,L),Y_F)&
                            +LATTICE(X_F,WRAP(Y_F-1,L)) + LATTICE(X_F,WRAP(Y_F+1,L)))
            ! Calculate spin-field energy change in non-zero field
            if (abs(G) >= 1E-6) then
                E_DIFF = E_DIFF - SUM(LATTICE)
            end if

            ! Accept/Reject Config
            if (E_DIFF > 0) then
                ! Reject with 1 - P(Boltzmann)
                call RANDOM_NUMBER(R_CA)
                if (exp(-E_DIFF/T) <= R_CA) then
                    !write(3,*) "POS REJECT", E_DIFF,R_CA, exp(-E_DIFF/T)

                    LATTICE(X_F,Y_F) = S
                    E_DIFF = 0.0
                else
                    !write(3,*) "POS ACCEPT", E_DIFF,R_CA, exp(-E_DIFF/T)
                end if
            else
                    !write(3,*) "NEG ACCEPT", E_DIFF,R_CA, exp(-E_DIFF/T)
            end if

            ! Update energy
            E = E+E_DIFF

            !call PRINT_LATTICE(LATTICE, 3)
            !write(3,*) E
            !call PRINT_OBSERVABLES(TIME,LATTICE,E,3)
            !write(3,*) "-------------------------"
        end do

        ! Calculate and accumulate observables

        E_SQ = E**2
        M = sum(LATTICE)
        M_SQ = M**2

        MEAN_E    = MEAN_E + E
        MEAN_M    = MEAN_M + M
        MEAN_E_SQ = MEAN_E_SQ + E_SQ
        MEAN_M_SQ = MEAN_M_SQ + M_SQ

        ! Data and debug Outputs

        !call PRINT_OBSERVABLES(TIME,E,E_SQ,M,M_SQ,2)

        !write(3,*) "-------------------------"
        !write(3,*) "AFTER TIME", TIME
```

```
            !write(3,*)
            !call PRINT_LATTICE(LATTICE,3)
            !call PRINT_OBSERVABLES(TIME,E,E_SQ,M,M_SQ,M_S,3)
            !write(3,*) "------------------------"

        end do

        ! Average out observables and normalize to lattice size

        MEAN_E      = MEAN_E    / EQ_SAMPLES
        MEAN_E_SQ   = MEAN_E_SQ / EQ_SAMPLES
        MEAN_M      = MEAN_M    / EQ_SAMPLES
        MEAN_M_SQ   = MEAN_M_SQ / EQ_SAMPLES

        C_V = MEAN_E_SQ - MEAN_E**2
        CHI = MEAN_M_SQ - MEAN_M**2

        CHI = CHI/size(LATTICE)
        C_V = C_V/size(LATTICE)

        MEAN_E = MEAN_E/size(LATTICE)
        MEAN_M = MEAN_M/size(LATTICE)

        ! Output to log and standard out

        write(1,103) "T = ", T
        write(1,103) "CHI_S =", CHI
        write(1,103) "C_V_S =", C_V
        write(1,103) "----------------------------"

        write(*,102) T, CHI, C_V, MEAN_E, MEAN_M
        !write(*,*) T, CHI,C_V,E/EQ_SAMPLES,M/EQ_SAMPLES

        T = T+T_STEP

        if (T>T_STOP) stop

    end do

    ! Format : Output
    ! ------------------------------------------
    101 format (A1,1X,A8,4X,5(A20,4X))      ! Table header
    106 format (A1,1X,5(A20,4X))            ! Table header
    102 format (2X,5(F20.10,4X))            ! Average Observables
    103 format (A15,F15.10)                 ! Single variables
    104 format (A15,I8)

end program ISING_MODEL
```

Output:

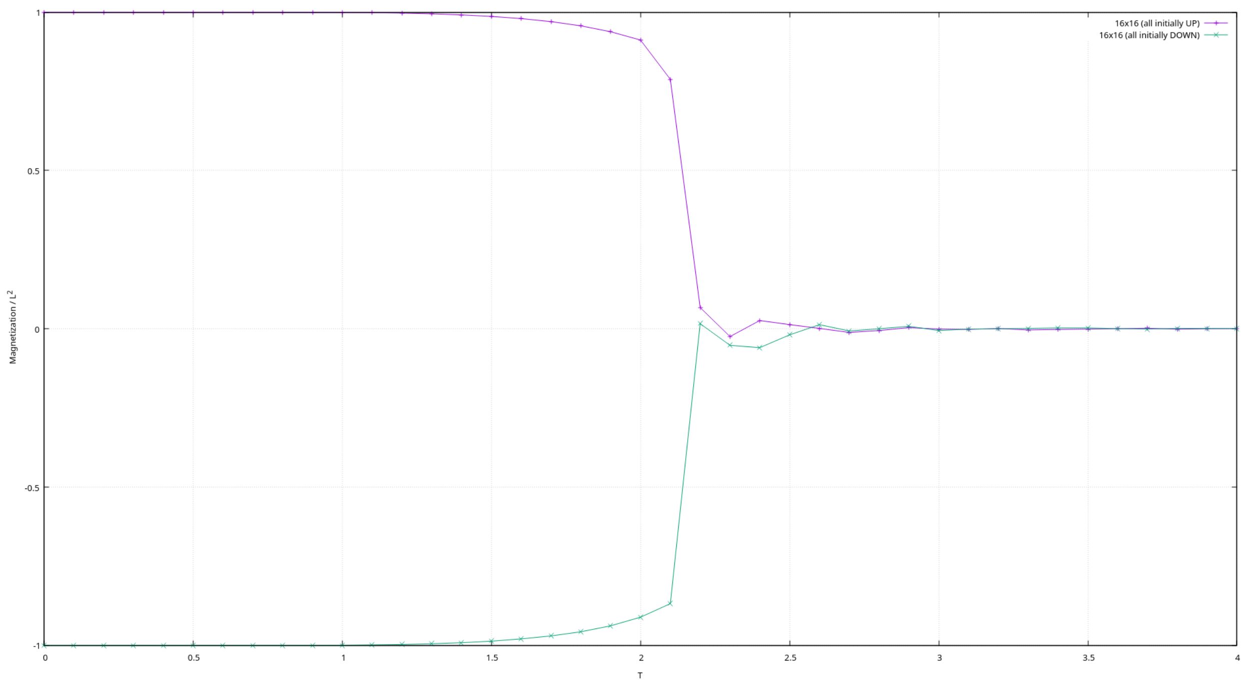


Figure 6: Magnetization per spin

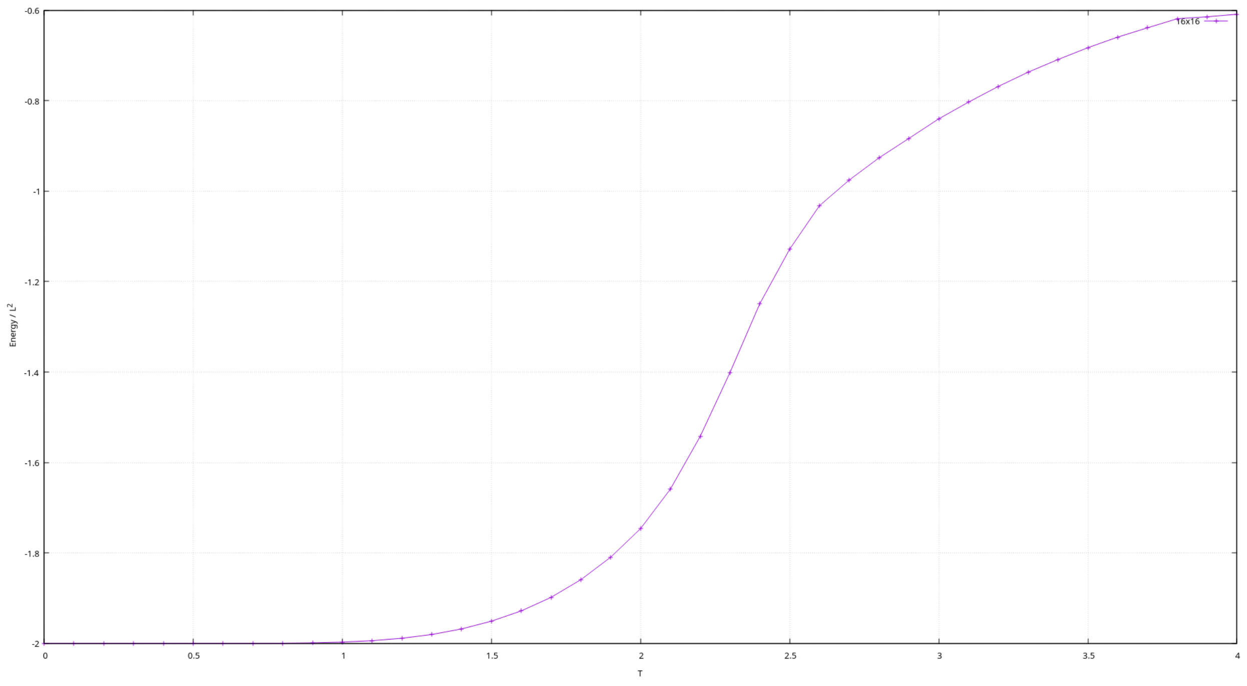


Figure 7: Energy per spin

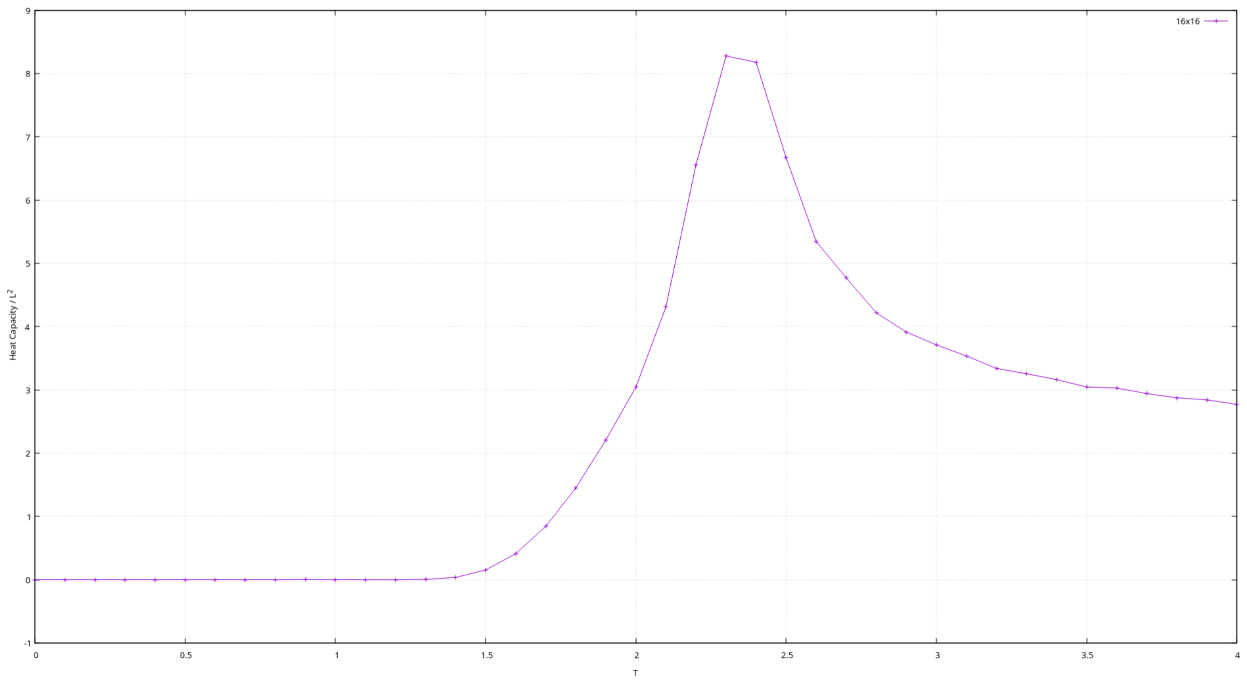


Figure 8: Heat Capacity per spin

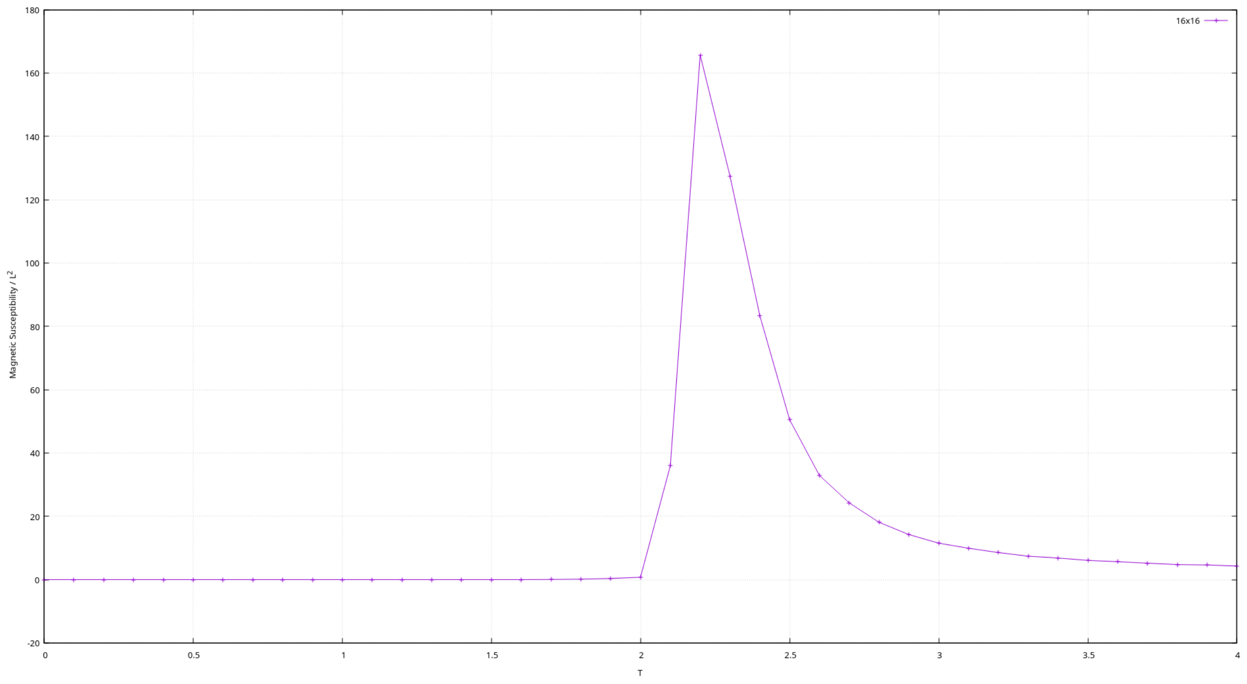


Figure 9: Magnetic Susceptibility per spin

APPENDIX - Common Subroutines & Functions

FORMULA

```
module FORMULA

    use iso_fortran_env, only: IK => int32, RK => real64

    implicit none
    public::NBALL_VOLUME
    double precision,parameter::PI = 4*atan(1.d0)

    contains

    real function NBALL_VOLUME(N,R)
        ! Compute the volume of a N-dimensional hypersphere of radius R

        integer,intent(in)::N
        real,intent(in)::R

        NBALL_VOLUME = (R**N)*(PI**(0.5*N))/gamma((0.5*N)+1)

    end function NBALL_VOLUME

    pure function GAUSSIAN(X,K) result(FUNC)

        real,dimension(:),intent(in)::X
        real,intent(in)::K
        real,dimension(SIZE(X))::FUNC

        FUNC(:) = exp(-K*(X(:)**2))

    end function GAUSSIAN

end module FORMULA
```

Flowcharts:

INTEGRATE

```fortran
module INTEGRATE

    use iso_fortran_env, only: IK => int32, RK => real64

    implicit none
    public::TRAPEZOID
    public::SIMPSON

contains

    subroutine TRAPEZOID(F,A,B,N,INTEGRAL)
        ! Composite Trapezoidal Rule
        !
        ! Integrates function F on [A,B] with N subintervals

        implicit none

        real, external      :: F            ! Integrand
        real, intent(in)    :: A            ! Interval start
        real, intent(in)    :: B            ! Interval end
        integer, intent(in) :: N            ! No. of subintervals
        real, intent(out)   :: INTEGRAL     ! Result

        integer :: I
        real    :: H

        H = (B-A)/N

        INTEGRAL = 0.5*(F(A) + F(B))

        do I = 1,N-1
            INTEGRAL = INTEGRAL + F(A+H*I)
        end do

        INTEGRAL = H*INTEGRAL

    end subroutine TRAPEZOID


    subroutine SIMPSON(F,A,B,N,INTEGRAL)
        ! Composite Simpson's 1/3rd Rule
        !
        ! Integrates function F on [A,B] with N subintervals

        implicit none

        real, external      :: F            ! Integrand
        real, intent(in)    :: A            ! Interval start
        real, intent(in)    :: B            ! Interval end
        integer, intent(in) :: N            ! No. of subintervals
        real, intent(out)   :: INTEGRAL     ! Result
        integer         :: I
        real            :: H
        real, parameter :: R = 4.0/3.0

        H = (B-A)/N
        write(*,*) R

        INTEGRAL = (F(A) + F(B))/3

        do I = 1,N-1,2
            INTEGRAL = INTEGRAL + R*F(A+H*I)
        end do

        do I = 2,N-1,2
            INTEGRAL = INTEGRAL + 0.5*R*F(A+H*I)
        end do

        INTEGRAL = H*INTEGRAL

    end subroutine SIMPSON


end module INTEGRATE
```

Flowcharts:

MONTECARLO

```
module MONTECARLO

    use iso_fortran_env, only: IK => int32, RK => real64

    implicit none
    public::ACCREJ_SPHERE

    contains

    subroutine ACCREJ_SPHERE(RAD,DIMM,NRAND,ESTVOL)
        ! Compute volume of DIMM-dimensional hypersphere of radius RAD
        ! by acceptance-rejection method

        implicit none

        real,intent(in)::RAD        ! Scaling factor
        integer,intent(in)::DIMM    ! Dimension of RN vector
        integer,intent(in)::NRAND   ! Number of throws
        real,intent(out)::ESTVOL    ! Estimated Volume

        integer::I
        real::NIN
        double precision,allocatable::R(:,:)

        ! Generate and shift RN vector
        allocate(R(NRAND,DIMM))
        call RANDOM_NUMBER(R)
        R = (R - 0.5)*(2*RAD)

        ! Count internal points
        NIN = 0
        do I=1,NRAND
            if (sum(R(I,:)**2) <= RAD**DIMM) then
                NIN = NIN+1
            endif
        end do

        ! Compute volume
        ESTVOL = (NIN/NRAND)*(2**DIMM)

        deallocate(R)

    end subroutine ACCREJ_SPHERE

end module MONTECARLO
```

Flowcharts: