PHY453: Computational Physics | Assignment 2

Anantha Rao | Reg no 20181044

15 Sept 2021

Contents

1 Generate random numbers

1.1 Observations

- Code for this section is available at `20181044_p1_testrdnnumbers.f90` and is depicted is Section-4. The compiled file is available at `20181044_p1.x`.
- **Allowed range of x**: The range of x is $(0, \infty)$ (please check figure 2 for derivation)
- The mean and standard deviation are: $(\mu = 1/2)$ and $(\sigma = 1/2)$ (please check figure 2 for derivation)

1.2 Random Number Testing

Observations::

1. The histogram for different number of random numbers shows an exponential distribution (e^{-x}), just like what we expect $(2e^{-2x})$
2. The calculations of mean and standard deviation yield the analytical results around (0.5, 0,5). We have the following results and observe that as we increase the number of random numbers, the accuracy increases!

Random numbers	Mean	Standard deviation
100	0.4072	0.4515
10000	0.4936	0.4986
1000000	0.4999	0.501

3. The autocorrelation plot usually hovers around 0 but peaks for large values of 'k'. This might be due to the limited size of the array. The scatter plot depicts a uniform distribution of points that resembles the probability distribution of interest ie $p(y)dy = 2exp(-2y)dy$

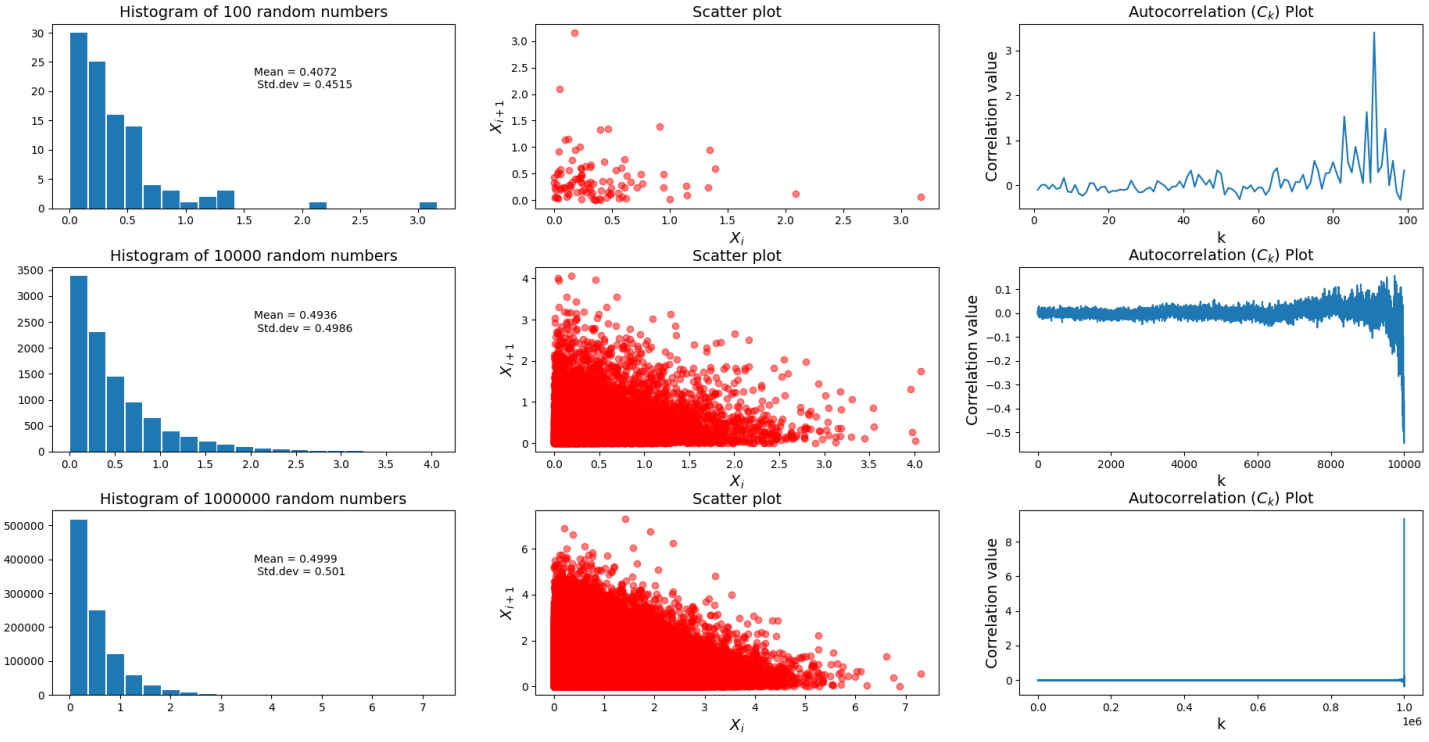

Figure 1: Random Number testing results

20181044

Q.1) Given distribution of interest $p(y)dy = e^{-2y}dy$

After normalisation on $y \in [0,\infty)$ we get

$p(y)dy = 2e^{-2y}dy$

Since Fortran 95 produces random numbers from the uniform distribution between $[0,1]$, we need to transform the called random variable to our distribution of interest. By conservation of probability, we have:

$p(y)dy = p(x)dx$ where $p(x) = \begin{cases} 1, & x \in [0,1] \\ 0, & \text{otherwise} \end{cases}$

$2e^{-2y'}dy' = 1 \cdot dx$

$\int_0^x dx' = 2\int_0^y e^{-2y'}dy'$

$x = 2\,\frac{e^{-2y'}}{-2}\Big|_0^y = 1 - e^{-2y}$

Thus we get $\boxed{y(x) = -\frac{1}{2}\log_e(1-x).}$

So given 'x' from the uniform distribution between $[0,1]$, we can get samples from $p(y)$ by using this relation

$x \in [0,1]$

$y \in [y(x=0), y(x=1)]$

$\Rightarrow \boxed{y \in [0,\infty)}$ Range of 'y' random variable

Finding Mean and standard deviation:

$\langle y \rangle = \int_0^\infty y(2e^{-2y})dy = 2\int_0^\infty y e^{-2y}dy$ (integration by parts)

$= 2\left[\frac{ye^{-2y}}{-2}\Big|_0^\infty - \frac{e^{-2y}}{4}\Big|_0^\infty\right] = 1/2$

$\langle y^2 \rangle = \int_0^\infty y^2(2e^{-2y})dy = 2\int_0^\infty y^2 e^{-2y}dy$

$= 2\left[\frac{-y^2e^{-2y}}{2}\Big|_0^\infty - \frac{ye^{-2y}}{2}\Big|_0^\infty - \frac{e^{-2y}}{4}\Big|_0^\infty\right] = \frac{1}{2}$

$\sigma_y^2 = \langle y^2 \rangle - \langle y \rangle^2$

$= 1/2 - 1/4 = 1/4$

$\sigma_y = 1/2$

So we have:

Mean: $\langle y \rangle = 1/2$

Standard deviation: $\sigma_y = 1/2$

Figure 2: Derivation of Mean and standard deviation for the distribution $p(y)dy = e^{-2y}dy$

2 One dimensional integral

- Code for this section is available at `20181044_p2_trap.f90`, `20181044_p2_hitmiss.f90`, `20181044_p2_impsampling.f90` and is depicted is Section-4. The compiled file is available with the same name and extension `.x`.
- Here we compute the value of $\int_0^3 e^x dx$ by three methods:

1. Trapezoidal method
2. Acceptance Rejection method
3. Importance sampling method

2.1 Results

1. Trapezoidal method (results stored in `20181044_p2_trap_e3.dat`)

No of Grid points	Computed Value	Absolute Error
1	31.628305384781502	12.542768461593834
10	19.228464196886080	0.1429272369841174
100	19.086968316986134	1.4313937984660186E-003
1000	19.085551237338208	1.4314150540428727E-005
10000	19.085537066329156	1.4314148799599025E-007
100000	19.085536924618982	1.4313137342014670E-009
1000000	19.085536923201797	1.4129142300589592E-011
10000000	19.085536923185874	1.7941204077942530E-012
100000000	19.085536923191025	3.3573144264664734E-012
1000000000	19.085536923191409	3.7410075037769275E-012

2. Acceptance Rejection method (results stored in `20181044_exp_accep_rej.dat`)

Random Numbers	Value of Integral	Absolute Error
1	0.0000000000000000	19.085536923187668
10	18.076983979423176	1.0085529437644922
100	11.448755921705697	7.6367810014819710
1000	19.523141979462252	0.43760505627458457
10000	18.788011453267245	0.29752546992042284
100000	18.946486059426206	0.13905086376146158
1000000	19.070434865816196	1.5102057371471744E-002
10000000	19.087794738195612	2.2578150079439752E-003
100000000	19.091036133627995	5.4992104403268627E-003

3. Importance Sampling method (results stored in `20181044_p2_MC_sampling.dat`)

Random Numbers	Value of Integral	Absolute Error
1	18.233492109562388	0.0000000000000000
10	18.691577856988648	0.94578377484162934
100	18.589300862800378	0.26514112074352941
1000	18.916875999346182	0.27274753496921239
10000	19.271068880725629	0.25460024260192826
100000	18.976531390678886	3.9316242708983463E-002
1000000	19.095807360623056	4.3481136876137756E-002
10000000	19.073275195810417	9.7157479944118500E-003
100000000	19.076293838257548	3.3547700428205852E-003

Here, we sample from the distribution $p(y) = \frac{1}{9}y^2 dy$ where $y \in (0, 3)$. We use the uniform distribution p(x) in (0,1)

and transform it to the desired sampling range by using $y = 3x^{1/3}$

2.2 Observations

1. We notice that in all the methods, the error decreases on increasing the number of grid points/random numbers.
2. The absolute error decreases rapidly in the trapezoidal method whereas it is converges slowly using the MC method. This is understood from the fact that the former has a $O(N^{-2})$ complexity whereas the later has a $O(N^{-1/2})$ complexity. Moreover, the standard deviation in important sampling methods depends substantially on the underlying distribution where we sample from. Here we use $p(x)dx = \frac{1}{9}x^2dx$ whereas other distributions could have converged faster.

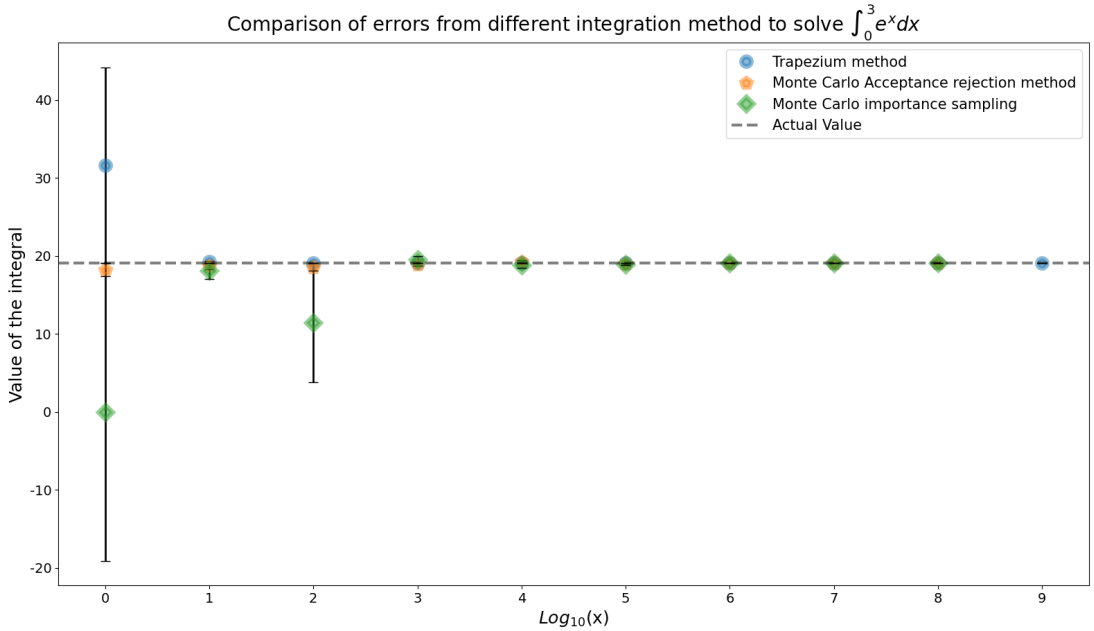

Figure 3: Comparisson of Results from different methods

3 6D Integral

- Code for this section is available at `20181044_p3_mc_bruteforce.f90`, `20181044_p3_mc_sampling.f90` and is depicted is Section-4. The compiled file is available with the same name and extension `.x`.
- Here we compute the multidimensional integral:

$$I = \int_{-\infty}^{\infty} g(\vec{x}, \vec{y}) d\vec{x} d\vec{y}$$

where

$$g(\vec{x}, \vec{y}) = exp(\vec{x}^2 - \vec{y}^2 - 0.5(\vec{x} - \vec{y})^2)$$

Here we compute the value of the integral by two methods:

- Brute Force Monte-Carlo integration
- Importance sampling Monte-Carlo integration

3.1 Results and Conclusions

1. Monte-Carlo integration by Brute force sampling (written in `20181044_multi_brute_mc.dat`)

Random Numbers	Actual Value	Standard deviaton
1	1.0037306924412803E-020	0.0000000000000000
10	1.9992274030063054E-012	1.8890121884805878E-012
100	7.4601672323693907E-007	6.5011226950469110E-007
1000	21.150726911349768	14.920730056556410
10000	12.371141249696983	7.8491229377295548
100000	13.400594409745494	3.4395903524405829
1000000	10.431964330001456	1.0452090281506743
10000000	11.479949789404076	0.40194791808710167
100000000	11.088242876799791	0.11841214528642795

2. Monte-Carlo integration by Importance sampling (written in `20181044_multi_impsampling_mc.dat`)

Random Numbers	Actual Value	Standard deviaton
1	1.7511242582255069	0.0000000000000000
10	7.7915233672702486	1.9398168362081580
100	10.613316207762105	0.70570929783960790
1000	10.773751805484222	0.25155026522297413
10000	10.936704202262902	8.0414454327716031E-002
100000	10.963500698020123	2.5488623564810239E-002
1000000	10.972169255486866	8.0498884359627640E-003
10000000	10.960832826817560	2.5467104189081458E-003
100000000	10.963558243628714	8.0528824856377311E-004

Since we do not know the exact value of the integral, we use the **standard deviation** (σ) from the MC integration technique to compare the efficiency of the two methods. We observe that, in the brute force method the lowest value of σ is around 0.11 which is around 1% of the integral value. However, the importance sampling method converges rapidly as $\frac{1}{\sqrt{N}}$ and errors atleast 4 orders lesser than the computed value. This demonstrates that although the later is a slightly expensive, it offers a better estimate of the value of the integral.

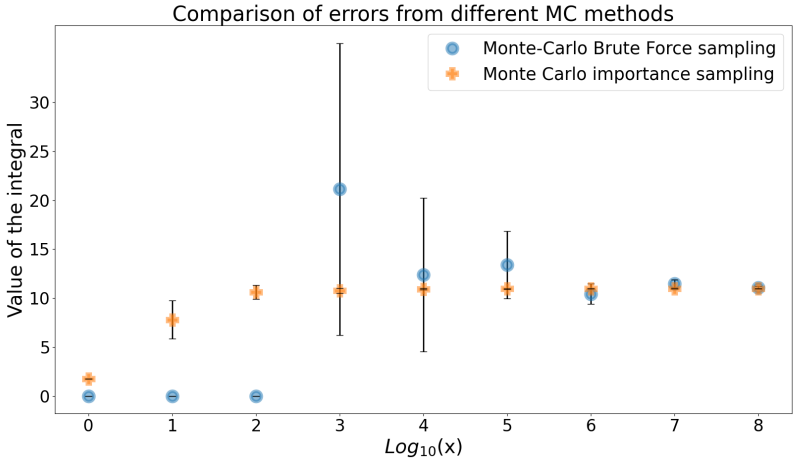

Figure 4: Comparisson of the results from the two MonteCarlo methods

4 Full code

```fortran
program test_randomnumbers
      ! This program generates random numbers from the distribution 2exp(-2x)dx
      ! And tests the (mean,std.dev), autocorrelation and the scatter-plot.
      ! Author : Anantha S Rao (Reg no:20181044)
      ! Date : 15 September 2021

      implicit none
      integer:: nbins, k, i, j
      ! nbins : Number of bins
      ! i, j, k : counters

      real*8:: p, xavg, xsqr_avg,sigma, x_ik
      ! p: Dummy variable to store random numbers
      ! xavg: store the averages value of the random variable (rv)
      ! xsqr_avg: store the average value of the square of rv
      ! x_ik : compute correlation between (i,k) indices

      real*8, dimension(:), allocatable::index_array,rdn_array,ck_array,ck
      ! index_array : Stores the indices 1:nbins
      ! rdn_array : Stores random numbers from the distribution
      ! ck_array : Stores correlation between rdn numbers
      ! ck : Stores autocorrelation between rdn numbers

      nbins = 10000
      allocate(rdn_array(nbins))
      allocate(index_array(nbins))
      allocate(ck_array(nbins-1))
      allocate(ck(nbins-1))

      ! Compute index array
      do i=1,nbins-1
        index_array(i)=i
      end do

      ! Compute random samples from the distribution
      ! 2exp(-2x)dx. We use -0.5log(x) for the same
      do i=1,nbins
        call random_number(p)
        rdn_array(i) = -0.50d0*log(p)
      end do

      ! We write the array of random numbers in a file
      open(unit=5, file="p1_randomnumbers.dat")
      do i=1,nbins
        write(5,*) rdn_array(i)
      end do
      print*,"Random numbers written to p1_randomnumbers.dat "

      ! Find mean and standard deviation
      xavg=0
      xsqr_avg=0
      do i=1,nbins
        xavg = xavg+rdn_array(i)
        xsqr_avg = xsqr_avg+(rdn_array(i)*rdn_array(i))
      end do
      xavg = xavg/real(nbins)
      xsqr_avg = xsqr_avg/real(nbins)

      sigma=sqrt(xsqr_avg - (xavg*xavg))

      write(*,*) "The mean is =", xavg
```

Figure 5: Fortran for Random number testing

```fortran
program trapezoidal_with_different_n
    ! Author : Anantha Rao (Reg no: 20181044)
    ! Program to compute the Integral f(x)=exp(x) with different number of bin sizes in powers of 10
      implicit none
      integer:: n,i
      ! n : number of bins
      ! i : counter

      real*8::a,b,h,func, fa,fb,trap_summ, error, actual_value
      ! a : Lower limit of the integral
      ! b : Upper limit of the integral
      ! h : Bin size
      ! trap_summ : Value of the integral using trapezoidal technique
      ! error : Value of the error between computed and analytical value
      ! actual_value : Analytical value of the integral


      open(unit=7, file='20181044_trap_e3.dat')
      n = 1
      a = 0
      b = 3
      actual_value = exp(3.0d0)-1

      print *,"Welcome to this program.&
          & This program does integration on I=0^3 exp(x)dx using trapezoidal rule on different
number of bins (n)"

      do
        h=(b-a)/real(n)
        write(*,10) n
        10 format("Computing for n=",i10)

        fa = func(a)/2.0d0
        fb = func(b)/2.0d0
        trap_summ=0.0d0

        do i=1, n-1
            trap_summ = trap_summ+func(a+h*i)
        end do

      trap_summ=(trap_summ+fa+fb)*h
      error=ABS(actual_value - trap_summ)
      write(7,*) n,"","",trap_summ,"",error
      if (n .ge. 1000000000) exit
      n=n*10
      end do
      print*, "Done!, Output stored in 20181044_trap_e^3.dat"

END PROGRAM

real*8 function func(x)
      !evaluates f(x)=e^x
      implicit none
      real*8::x
      func=exp(x)
end function
```

Figure 6: Fortran code for Trapezium integral

```
program hitmiss_mc_int
      ! Program to evaluate I= 0^3 exp(3)dx using the Monte-Carlo Hit-or-Miss technique

      implicit none
      integer:: counter, nbins, npts_in_curve
      ! counter : counter variable
      ! nbins : number of bins
      ! npts_in_curve : number of points that lie within the region of interest

      real*8:: x, p, y, integral, actual_value
      ! x : varialbe storing random number of x-axis
      ! y : varialbe storing random number of y-axis
      ! p : dummy variable for the random number generator
      ! integral : value of the computed integral
      ! actual_value : Analytic value of the integral

     print *,"Welcome to this program.&
        & This program does integration on I=0^3 exp(x)dx using Monte-Carlo Hit & Miss method on
different number of bins (n) "

      nbins=1
      actual_value=exp(3.0d0)-1
      7 npts_in_curve=0
      open(unit=1, file="exp_accep_rej.dat")
      write(*,10) nbins
      10 format("Computing for nbins=", i10)

      do counter=1,nbins
          ! find random value of "x" between 0 and 3
          call random_number(p)
          x=3.0d0*p

          ! find value of function between 0 and exp(3)
          call random_number(p)
          y=exp(3.0)*p

          ! If y at exp(x) is below the curve we accept the number
          if (y .lt. exp(x)) then
              npts_in_curve=npts_in_curve+1
          end if
      end do

      !multiply with area of rectangle and divide by the no. of cycles
      integral=3.0d0*exp(3.0)*(real(npts_in_curve)/real(nbins))
      write(1,*) nbins,"",integral,"",abs(integral-actual_value)

      ! automated
      nbins=nbins*10
      if (nbins .le. 100000000 ) goto 7
      print*, "Done! Output stored in exp_accep_rej.dat"
      end program hitmiss_mc_int
```

Figure 7: Fortran code for Monte-Carlo Hit and Miss technique for integration

```
program multi_dimensional_bruteforce_mc
      ! This program computes the 6D integral by Brute-force MonteCarlo sampling
      ! Author : Anantha Rao
      ! Reg no : 20181044
      ! 6D (x1,x2,x3,x4,x5,x6) Integral
      implicit none
      integer:: n,i,j
      ! n : no of bins
      ! i,j : counters
      real*8 :: x(6), y, func, int_mc, var, sigma, p(6)
      ! p : Dummy argument for random number generator (6 values)
      ! x(6), p(6) : vectors that stores the x-vector and random-variable respectively
      ! func : function that is the integrand
      ! int_mc : Value of the integrand
      ! var : variance
      ! sigma : standard deviation used to compute the accuracy of the estimate

      real*8 :: length, volume
      length=5.0d0 ! use one side of each dimension (0,5)
      volume=(2.0d0*length)**6  ! volume of 6D space

      open(unit=1, file="multi_brute_mc.dat")
      n=1

print *,"Welcome to this program.&
& This program solves mutidimensional integral using Monte-Carlo rule"

    7 int_mc=0.0d0
      var=0.0d0
      sigma=0.0d0
      write(*,10) n
      10 format("Computig for n=",i10)

      ! Compute integral
      do i=1,n
        call random_number(p)
        do j=1,6
            x(j) = -length + 2.0d0*length*p(j)
        end do
        int_mc=int_mc+func(x)
        sigma=sigma+func(x)*func(x)
      end do

      int_mc = int_mc/real(n)
      sigma=sigma/real(n)
      var=sigma-int_mc*int_mc

      int_mc = volume*int_mc
      sigma=volume*sqrt(var/real(n))

      write(1,*) n, "", int_mc, "", sigma

      ! begin automation
      n=n*10
      if (n .lt. 1000000000 ) goto 7
      print*,"Done! Output stored in file multi_brute_mc.dat"
      ! end automation

end program

real*8 function func(x)
      implicit none
      real*8::x(6), xx, yy, xy
      real*8::a,b
      a=1.0d0
      b=0.5d0

      xx= x(1)*x(1) + x(2)*x(2) + x(3)*x(3)
      yy= x(4)*x(4) + x(5)*x(5) + x(6)*x(6)
      xy= (x(1)-x(4))**2 + (x(2)-x(5))**2 + (x(3)-x(6))**2
      func=exp(-a*xx - a*yy - b*xy)

end function
```

Figure 8: Fortran code to compute the 6D integral by Brute Force monte-Carlo simulation

```
program multi_dimensional_bruteforce_mc
      ! This program computes the 6D integral by Brute-force MonteCarlo importance sampling
      ! Author : Anantha Rao
      ! Reg no : 20181044
      ! 6D (x1,x2,x3,x4,x5,x6) Integral
      implicit none
      integer:: n,i,j
      real*8 :: x(6), y, func, int_mc, var, sigma, p(2)
      ! p : Dummy argument for random number generator (2 values)
      real*8 :: length, volume, sqrt2, gauss_dev

      length=5.0d0 ! use one side of each dimension (0,5)
      volume=acos(-1.0d0)**3  ! volume of 6D space
      sqrt2=1.0d0/sqrt(2.0d0)

      open(unit=1, file="multi_impsampling_mc.dat")
      n=1

print *,"Welcome to this program.&
& This program solves mutidimensional integral using Monte-Carlo rule with sampling"

    7 int_mc=0.0d0
      var=0.0d0
      sigma=0.0d0
      write(*,10) n
      10 format("Computig for n=",i10)

      do i=1,n
        do j=1,6
            call random_number(p)
            x(j) = gauss_dev(p)*sqrt2
        end do
        int_mc=int_mc+func(x)
        sigma=sigma+func(x)*func(x)
      end do

      int_mc = int_mc/real(n)
      sigma=sigma/real(n)
      var=sigma-int_mc*int_mc

      int_mc = volume*int_mc
      sigma=volume*sqrt(var/real(n))

      write(1,*) n, "", int_mc, "", sigma

      ! begin automation
      n=n*10
      if (n .lt. 1000000000 ) goto 7
      print*,"Done! Output stored in file multi_impsampling_mc.dat"
      ! end automation
end program multi_dimensional_bruteforce_mc

real*8 function func(x)
      implicit none
      real*8::x(6), xy, a
      a=0.5d0

      xy= (x(1)-x(4))**2 + (x(2)-x(5))**2 + (x(3)-x(6))**2
      func=exp(- a*xy)

end function

real*8 function gauss_dev(x)
    implicit none
    real*8:: fact, sqr, p, x1, x2, x(2)

    7 call random_number(p)
    x1 = 2.0d0*p - 1.0d0
    call random_number(p)
    x2 = 2.0d0*p - 1.0d0
    sqr = x1*x1 + x2*x2

    if (sqr .ge. 1.0d0 .or. sqr .eq. 0.0d0) goto 7

    fact=sqrt(-2.0d0*log(sqr)/sqr)
    gauss_dev=x2*fact
    end function
```

Figure 9: Fortran code to compute the 6D integral by Monte-Carlo importance sampling