

**Problem 1 (Sample Mean Method. (5 points))** Score: \_\_\_\_\_. Alternative to the *Hit or Miss* algorithm, one can calculate an integral from the mean-value theorem of calculus.

$$F = \int_a^b dx f(x) = (b-a)\langle f \rangle. \quad (1)$$

$\langle f \rangle$  is the average value of the function  $f(x)$  in the range  $a \leq x \leq b$ . The sampled mean value method estimates the average  $\langle f \rangle$  as follows:

- (a) We choose  $n$  random number  $x_i$  from the interval  $[a, b]$ , which are distributed uniformly.
- (b) We compute the values of the function  $f(x)$  at these points.
- (c) We take their average  $\langle f \rangle = \frac{1}{n} \sum_{i=1}^n f(x_i)$ . Please implement the above method parallelly and evaluate  $\int_0^3 e^x dx$  as a function of  $n$ .

**Solution:** 用sample mean算法计算 $F$ , 其中抽取的数据点数量 $n$ 从1000以1000的步长递增到1000000, Fortran代码见附录, 积分结果 $F$ 随抽取的数据点数量 $n$ 的变化情况如图1.

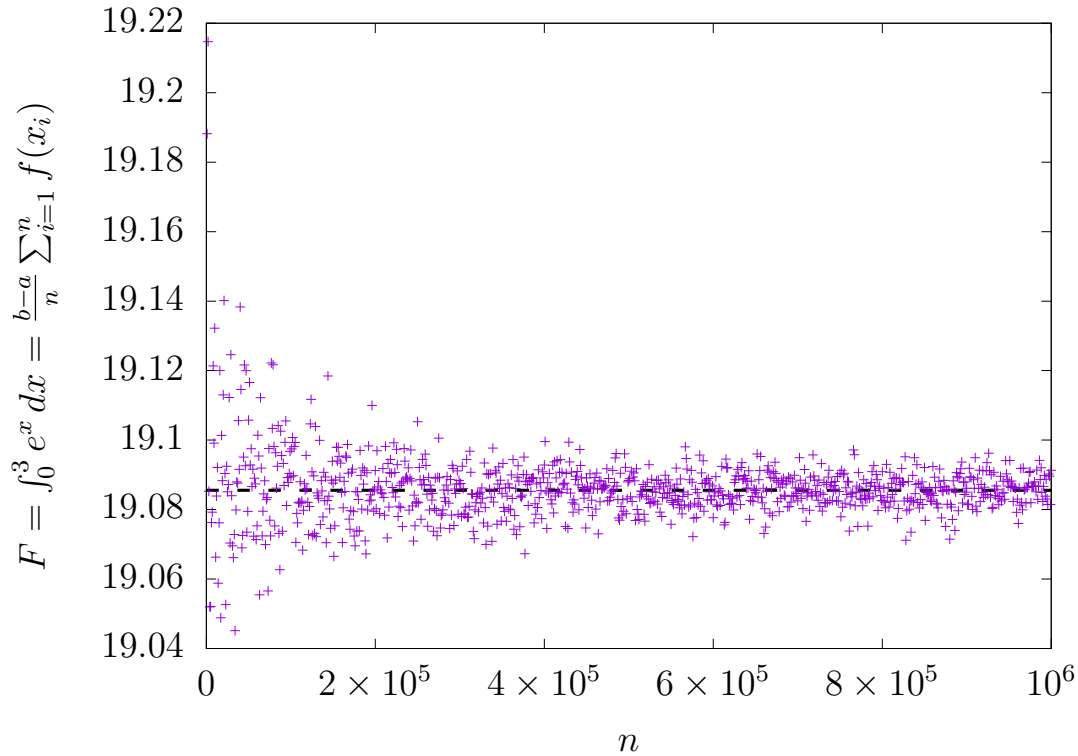


图 1: 积分结果 $F$ 随抽取的数据点数量 $n$ 的变化情况, 其中的黑色虚线是积分的准确值.

由图1可见, 随着抽取的数据点数量 $n$ 增加, Monte Carlo积分结果越来越接近于积分的准确值, 这符合中心极限定理. □

**Problem 2 (Sample Mean method for higher dimensional integration. (5 points))** Score: \_\_\_\_\_. Please perform the following three dimensional integration with two different methods: one uses the conventional integration on equally discretized points, the other one uses the Monte Carlo method. By using the same number of points in the two methods, compare the accuracy and speed of the two algorithms.

$$\int_0^{2\pi} d\phi \int_0^{3a} \rho^3 d\rho \int_{-\sqrt{9a^2-\rho^2}}^{\sqrt{9a^2-\rho^2}} dz = \frac{648\pi}{5} a^3. \quad (2)$$

**Solution:** 取  $a = 1$ .

- **传统数值积分方法:** 用拓展到三维的中矩形公式计算积分, 算法如下, 代码见附录.

1. 在  $\phi \in [0, 2\pi], \rho \in [0, 3a], z \in [-3a, 3a]$  范围内均匀取离散点, 其中  $\phi$  和  $z$  维度离散  $N$  份,  $\rho$  维度离散  $N \times N_{\text{tasks}}$  份 ( $N_{\text{tasks}}$  为参与运算的节点数量):

$$\phi_i = \left(i - \frac{1}{2}\right) \Delta\phi, \quad \Delta\phi = \frac{2\pi - 0}{500} \quad i = 1, 2, \dots, 500, \quad (3)$$

$$\rho_j = \left(j - \frac{1}{2}\right) \Delta\rho, \quad \Delta\rho = \frac{3a - 0}{500}, \quad j = 1, 2, \dots, 500, \quad (4)$$

$$z_k = -3a + \left(k - \frac{1}{2}\right) \Delta z, \quad \Delta z = \frac{3a - (-3a)}{500}, \quad k = 1, 2, \dots, 500. \quad (5)$$

2. 计算各个离散点上的函数值:

$$f(\phi_i, \rho_j, z_k) = \rho_j^3, \quad (6)$$

其中第  $n$  个节点计算标号为  $i = 1, 2, \dots, N, \quad j = 0 \times N_{\text{tasks}} + n, 1 \times N_{\text{tasks}} + n, \dots, (N-1)N_{\text{tasks}} + n, \quad k = 1, 2, \dots, N$  的离散点 (共  $N^3$  个) 上的函数值.

3. 判断各个离散点是否处于  $\phi \in [0, 2\pi], \rho \in [0, 3a], z \in [-\sqrt{9a^2 - \rho^2}, \sqrt{9a^2 - \rho^2}]$  这一积分范围内, 用指标  $H$  来表示:

$$H(\phi_i, \rho_j, z_k) = \begin{cases} 1, & \phi_i \in [0, 2\pi], \rho_j \in [0, 3a], z_k \in [-\sqrt{9a^2 - \rho_j^2}, \sqrt{9a^2 - \rho_j^2}], \\ 0, & \text{otherwise.} \end{cases} \quad (7)$$

4. 将所有处于积分范围内的离散点上的函数值求和取平均并乘上积分范围的体积, 得到数值积分结果:

$$I \approx \left(\frac{2\pi - 0}{N}\right) \left(\frac{3a - 0}{N}\right) \left(\frac{3a - (-3a)}{N}\right) \sum_{i=1}^N \sum_{j=1}^N \sum_{k=1}^N f(\phi_i, \rho_j, z_k) H(\phi_i, \rho_j, z_k). \quad (8)$$

5. 用 CPU\_TIME 来得到各个节点的计算耗时, 将所有节点的计算耗时相加得到计算总耗时, 将计算结果与积分的准确值  $\frac{648\pi}{5}a^3$  做差得到计算误差.

- **Monte Carlo 方法:**

– **Sample mean 方法:** 算法与前一题题干中针对一维情况的大体相同, 代码见附录

1.  $N_{\text{tasks}}$  个具有不同 seed 的节点参与运算, 每个节点在  $\phi \in [0, 2\pi], \rho \in [0, 3a], z \in [-3a, 3a]$  范围内均匀随机抽取  $N^3$  个点.
2. 计算各个随机抽取的点上的函数值:

$$f(\phi_{n,i}, \rho_{n,i}, z_{n,i}) = \rho_{n,i}^3. \quad (9)$$

其中  $n = 0, 1, \dots, N_{\text{tasks}} - 1$  代表 CPU 节点的序号,  $i = 1, 2, \dots, N$  代表随机抽取的点的序号.

3. 判断各个离散点是否处于  $\phi \in [0, 2\pi], \rho \in [0, 3a], z \in [-\sqrt{9a^2 - \rho^2}, \sqrt{9a^2 - \rho^2}]$  这一积分范围内, 用指标  $H$  来表示:

$$H(\phi_{n,i}, \rho_{n,i}, z_{n,i}) = \begin{cases} 1, & \phi_{n,i} \in [0, 2\pi], \rho_{n,i} \in [0, 3a], z_{n,i} \in [-\sqrt{9a^2 - \rho_{n,i}^2}, \sqrt{9a^2 - \rho_{n,i}^2}], \\ 0, & \text{otherwise.} \end{cases} \quad (10)$$

4. 将所有处于积分范围内的离散点上的函数值求和取平均并乘上积分范围的体积, 得到数值积分结果:

$$I \approx \frac{1}{N_{\text{tasks}}} \left(\frac{2\pi - 0}{N}\right) \left(\frac{3a - 0}{N}\right) \left(\frac{3a - (-3a)}{N}\right) \sum_{n=0}^{N_{\text{tasks}}-1} \sum_{i=1}^N f(\phi_{n,i}, \rho_{n,i}, z_{n,i}) H(\phi_{n,i}, \rho_{n,i}, z_{n,i}). \quad (11)$$

5. 用CPU\_TIME来得到各个节点的计算耗时，将所有节点的计算耗时加和来作为计算总耗时，以积分的准确值  $\frac{648\pi}{5}a^3$  为平均值计算所有节点计算结果的标准差来作为计算误差：

$$\sigma = \left[ \frac{1}{N_{\text{tasks}}} \sum_{n=0}^{N_{\text{tasks}}-1} \left( I_n - \frac{648\pi}{5}a^3 \right)^2 \right]^{1/2}. \quad (12)$$

其中

$$I_n = \left( \frac{2\pi}{N} \right) \left( \frac{3a-0}{N} \right) \left( \frac{3a-(-3a)}{N} \right) \sum_{i=1}^{N^3} f(\phi_{n,i}, \rho_{n,i}, z_{n,i}) H(\phi_{n,i}, \rho_{n,i}, z_{n,i}) \quad (13)$$

是标号为 $n$ 的节点计算得到的积分值。

– **Hit-or-Miss方法**：算法如下，代码见附录。

1.  $N_{\text{tasks}}$  个具有不同seed的节点参与运算，每个节点在  $\phi \in [0, 2\pi], \rho \in [0, 3\pi], z \in [-3a, 3a], f \in [0, 27a^3]$  范围内均匀随机抽取  $N^3$  个点。
2. 每个节点计算各自的  $N^3$  个点中满足

$$f_{n,i} < \rho_{n,i}^3, \quad (14)$$

并且在积分范围内

$$\phi_{n,i} \in [0, 2\pi], \quad \rho_{n,i} \in [0, 3a], \quad z_{n,i} \in [-\sqrt{9a^2 - \rho^2}, \sqrt{9a^2 - \rho^2}], \quad (15)$$

的点数  $n_{\text{in},n}$ 。

3. 用下式计算数值积分结果：

$$I \approx \frac{1}{N_{\text{tasks}}} \sum_{n=0}^{N_{\text{tasks}}-1} \frac{n_{\text{in},i}}{N} \left( \frac{2\pi-0}{N} \right) \left( \frac{3a-0}{N} \right) \left( \frac{3a-(-3a)}{N} \right) (27a^3 - 0). \quad (16)$$

4. 用CPU\_TIME来得到各个节点的计算耗时，将所有节点的计算耗时加和来作为计算总耗时，以积分的准确值  $\frac{648\pi}{5}a^3$  为平均值计算所有节点计算结果的标准差来作为计算误差：

$$\sigma = \left[ \frac{1}{N_{\text{tasks}}} \sum_{n=0}^{N_{\text{tasks}}-1} \left( I_n - \frac{648\pi}{5}a^3 \right)^2 \right]^{1/2}. \quad (17)$$

其中

$$I_n = \sum_{i=1}^{N^3} \frac{n_{\text{in},n}}{N} \left( \frac{2\pi-0}{N} \right) \left( \frac{3a-0}{N} \right) \left( \frac{3a-(-3a)}{N} \right) (27a^3 - 0). \quad (18)$$

统一选取  $a = 1, N = 500, n_{\text{tasks}} = 16$ （也就是说三种方法都是总共对  $500^3 \times 16$  个点进行了计算）来测试上述三种方法，其计算结果、总耗时、计算误差如表1。

表 1: 测试结果汇总。

选用的积分方法	计算结果	总耗时(s)	计算误差
传统数值积分方法	407.14869	40.91978	-0.00171
Sample mean方法	407.15133	342.42996	0.00532
Hit-or-Miss方法	407.12599	433.94705	0.01177

可见在积分只有三维的情况下，传统数值积分方法的速度和精度都尚优于Monte Carlo方法。其实仔细研究传统数值积分方法和sample mean算法的代码，可以发现两者的主要区别就在于选取离散点的方法，前者通过循环均匀遍历积分的区间，而后者通过随机数发生器随机抽取积分范围内的点，所以推测导致后者比前者耗时更多的原因是调用随机数发生

器会消耗较多的资源. Hit-or-Miss方法比sample mean方法更耗时间, 是因为后者只需要计算随机抽取的点上的函数值即可, 而前者不仅要计算随机抽取的点上的函数值, 而且还要将猜测的函数值与真实的函数值进行比较. Hit-or-Miss方法也比sample mean方法精度更差, 这是因为对于每个随机抽取的点, 后者精确地计算了该点的函数值, 而前者仅得到了猜测的函数值与真实的函数的大小关系这一信息, 显然是前者在模拟中获得的信息更多, 因而精度更高.

当然, 对于更高维度的积分, 传统数值积分方法的资源消耗可能会大幅增加, 这时可能Monte Carlo积分方法更为适用. □

## 附录

### Problem 1 sample mean算法Fortran代码

```

1  program main
2      use mpi
3      implicit none
4      integer :: ntasks, rank, ierr
5      integer, allocatable :: status(:)
6      integer :: clock, n
7      integer, allocatable :: seed(:)
8      integer(4) :: i, n_tot
9      real(8), parameter :: l = 0.d0, r = 3.d0
10     real(8) :: x, y
11     real(8) :: integral_local, integral
12
13     ! initialize the MPI environment
14     call MPLINIT(ierr)
15     call MPLCOMM_SIZE(MPLCOMM_WORLD, ntasks, ierr)
16     call MPLCOMM_RANK(MPLCOMM_WORLD, rank, ierr)
17     allocate(status(MPLSTATUS_SIZE))
18
19     ! initialize the random number for different processes
20     if (rank == 0) then
21         call SYSTEMCLOCK(clock)
22         call RANDOMSEED(size = n)
23         allocate(seed(n))
24         do i = 1, n
25             seed(i) = clock + 37 * i
26         end do
27         call RANDOMSEED(PUT = seed)
28         deallocate(seed)
29         do i = 1, ntasks - 1
30             call RANDOMNUMBER(x)
31             clock = clock + Int(x * 1000000)
32             call MPLSEND(clock, 1, MPLINTEGER, i, i, MPLCOMM_WORLD, ierr)
33         end do
34         open(unit = 1, file = '1-I-n.txt', status = 'unknown')

```

```
35     else
36         call MPLRECV(clock, 1, MPLINTEGER, 0, rank, MPLCOMMWORLD, status, ierr)
37         call RANDOMSEED(size = n)
38         allocate(seed(n))
39         do i = 1, n
40             seed(i) = clock + 37 * i
41         end do
42         call RANDOMSEED(PUT = seed)
43         deallocate(seed)
44     end if
45
46     do n_tot = 1000, 1000000, 1000
47         integral_local = 0.d0
48         integral = 0.d0
49         do i = 1, n_tot
50             call RANDOMNUMBER(x)
51             x = x * (r - 1) + 1
52             call func(x, y)
53             integral_local = integral_local + y
54         end do
55         integral_local = (r - 1) * integral_local / dble(n_tot)
56
57         call MPLREDUCE(integral_local, integral, 1, MPIREAL8, MPLSUM, 0,
58             MPLCOMMWORLD, ierr)
59
60         if (rank == 0) then
61             integral = integral / dble(ntasks)
62             write(1, '(i10,f10.5)') n_tot, integral
63         end if
64     end do
65
66     if (rank == 0) then
67         close(1)
68     end if
69
70     call MPIFINALIZE(ierr)
71 end program main
72
73 subroutine func(x, y)
74     ! the function to be integrated
75     implicit none
76     real(8), intent(in) :: x
77     real(8), intent(out) :: y
78
79     y = exp(x)
```

```
79 end subroutine func
```

### Problem 2 传统数值积分方法计算积分Fortran代码

```
1  program main
2  use mpi
3  implicit none
4  integer :: ntasks, rank, ierr
5  integer, allocatable :: status(:)
6  integer, parameter :: n = 500
7  integer :: i, j, k
8  real(8), parameter :: pi = acos(-1.d0)
9  real(8), parameter :: phi_l = 0.d0, phi_u = 2 * pi, rho_l = 0.d0, rho_u = 3.d0, z_l
   = -3.d0, z_u = 3.d0
10 real(8), parameter :: d_phi = (phi_u - phi_l) / dble(n), d_rho = (rho_u - rho_l) /
   dble(n), d_z = (z_u - z_l) / dble(n)
11 real(8) :: phi, rho, z, f, H
12 real(8) :: integral_local = 0.d0, integral
13 real :: start, finish, time
14
15
16 ! initialize the MPI environment
17 call MPIINIT(ierr)
18 call MPLCOMM_SIZE(MPLCOMM_WORLD, ntasks, ierr)
19 call MPLCOMM_RANK(MPLCOMM_WORLD, rank, ierr)
20 allocate(status(MPI_STATUS_SIZE))
21
22 call CPU_TIME(start)
23
24 phi = phi_l - d_phi / dble(2)
25 do i = 1, n
26     phi = phi + d_phi
27     rho = rho_l - d_rho + d_rho / dble(ntasks) / dble(2) + d_rho / dble(ntasks) *
       dble(rank)
28     do j = 1, n
29         rho = rho + d_rho
30         z = z_l - d_z / dble(2)
31         do k = 1, n
32             z = z + d_z
33             call func(phi, rho, z, f)
34             call inArea(phi, rho, z, H)
35             integral_local = integral_local + f * H
36         end do
37     end do
38 end do
39
```

```

40      call MPIREDUCE(integral_local , integral , 1, MPI_REAL8, MPLSUM, 0, MPI_COMM_WORLD,
41                    ierr)
42
43      if (rank == 0) then
44          integral = (phi_u - phi_l) * (rho_u - rho_l) * (z_u - z_l) / dble(ntasks) /
45                      dble(n**3) * integral
46          write(*, '(f10.5)') integral
47      end if
48
49      call CPU_TIME(finish)
50      call MPIREDUCE(finish - start , time , 1, MPI_REAL, MPLSUM, 0, MPI_COMM_WORLD, ierr)
51
52      if (rank == 0) then
53          write(*, '(f10.5)') time
54      end if
55
56      call MPI_FINALIZE(ierr)
57 end program main
58
59 subroutine func(phi, rho, z, f)
60     ! the function to be integrated
61     implicit none
62     real(8), intent(in) :: phi, rho, z
63     real(8), intent(out) :: f
64
65     f = rho**3
66 end subroutine func
67
68 subroutine inArea(phi, rho, z, H)
69     ! judge whether the dot is in the integral area
70     real(8), intent(in) :: phi, rho, z
71     real(8), intent(out) :: H
72
73     if ((z > -sqrt(9.d0 - rho**2)) .and. (z < sqrt(9.d0 - rho**2))) then
74         H = 1.d0
75     else
76         H = 0.d0
77     end if
78 end subroutine inArea

```

### Problem 2 sample mean 算法Fortran代码

```

1      program main
2      use mpi
3      implicit none
4      integer :: ntasks, rank, ierr

```

```

5  integer, allocatable :: status(:)
6  integer :: clock, n
7  integer, allocatable :: seed(:)
8  integer(4) :: i, n_tot = 125000000
9  real(8), parameter :: pi = acos(-1.d0)
10 real(8), parameter :: phi_l = 0.d0, phi_u = 2.d0 * pi, rho_l = 0.d0, rho_u = 3.d0,
    z_l = -3.d0, z_u = 3.d0
11 real(8) :: phi, rho, z, f, H
12 real(8) :: integral_local = 0.d0, integral
13 real :: start, finish, time
14 real(8) :: sigma
15
16 ! initialize the MPI environment
17 call MPIINIT(ierr)
18 call MPLCOMM_SIZE(MPLCOMM_WORLD, ntasks, ierr)
19 call MPLCOMM_RANK(MPLCOMM_WORLD, rank, ierr)
20 allocate(status(MPI_STATUS_SIZE))
21
22 call CPU_TIME(start)
23
24 ! initialize the random number for different processes
25 if (rank == 0) then
26     call SYSTEMCLOCK(clock)
27     call RANDOMSEED(size = n)
28     allocate(seed(n))
29     do i = 1, n
30         seed(i) = clock + 37 * i
31     end do
32     call RANDOMSEED(PUT = seed)
33     deallocate(seed)
34     do i = 1, ntasks - 1
35         call RANDOMNUMBER(z)
36         clock = clock + Int(z * 1000000)
37         call MPLSEND(clock, 1, MPLINTEGER, i, i, MPLCOMM_WORLD, ierr)
38     end do
39 else
40     call MPLRECV(clock, 1, MPLINTEGER, 0, rank, MPLCOMM_WORLD, status, ierr)
41     call RANDOMSEED(size = n)
42     allocate(seed(n))
43     do i = 1, n
44         seed(i) = clock + 37 * i
45     end do
46     call RANDOMSEED(PUT = seed)
47     deallocate(seed)
48 end if

```



```

49
50  do i = 1, n_tot
51      call RANDOMNUMBER(phi)
52      phi = phi * (phi_u - phi_l) + phi_l
53      call RANDOMNUMBER(rho)
54      rho = rho * (rho_u - rho_l) + rho_l
55      call RANDOMNUMBER(z)
56      z = z * (z_u - z_l) + z_l
57      call func(phi, rho, z, f)
58      call inArea(phi, rho, z, H)
59      integral_local = integral_local + f * H
60  end do
61  integral_local = (phi_u - phi_l) * (rho_u - rho_l) * (z_u - z_l) / dble(n_tot) *
    integral_local
62
63  call MPLREDUCE(integral_local, integral, 1, MPIREAL8, MPLSUM, 0, MPLCOMMWORLD,
    ierr)
64
65  if (rank == 0) then
66      integral = integral / dble(ntasks)
67      write(*, '(f10.5)') integral
68  end if
69
70  call CPU_TIME(finish)
71  call MPLREDUCE(finish - start, time, 1, MPIREAL, MPLSUM, 0, MPLCOMMWORLD, ierr
    )
72  if (rank == 0) then
73      write(*, '( "Time_consumed: ", f10.5)') time
74  end if
75  call MPLREDUCE((((integral_local - dble(648) * pi / dble(5))**2)**2, sigma, 1,
    MPIREAL8, MPLSUM, 0, MPLCOMMWORLD, ierr)
76  if (rank == 0) then
77      write(*, '( "Error: ", f10.5)') sqrt(sigma / dble(ntasks))
78  end if
79
80  call MPIFINALIZE(ierr)
81 end program main
82
83 subroutine func(phi, rho, z, f)
84     ! the function to be integrated
85     implicit none
86     real(8), intent(in) :: phi, rho, z
87     real(8), intent(out) :: f
88
89     f = rho**3

```

```

90 end subroutine func
91
92 subroutine inArea(phi, rho, z, H)
93     ! judge whether the dot is in the integral area
94     real(8), intent(in) :: phi, rho, z
95     real(8), intent(out) :: H
96
97     if ((z > -sqrt(9.d0 - rho**2)) .and. (z < sqrt(9.d0 - rho**2))) then
98         H = 1.d0
99     else
100         H = 0.d0
101     end if
102 end subroutine inArea

```

## Problem 2 hit-or-miss算法Fortran代码

```

1  program main
2  use mpi
3  implicit none
4  integer :: ntasks, rank, ierr
5  integer, allocatable :: status(:)
6  integer :: i, n, clock
7  integer, allocatable :: seed(:)
8  real(8), parameter :: pi = acos(-1.d0)
9  integer(4) :: n_in_local = 0, n_in, n_tot = 125000000
10 real(8), parameter :: phi_l = 0.d0, phi_u = 2 * pi, rho_l = 0.d0, rho_u = 3.d0, z_l
    = -3.d0, z_u = 3.d0, &
11     f_l = 0.d0, f_u = 27.d0
12 integer :: H
13 real(8) :: phi, rho, z, f, f_real
14 real(8) :: integral_local, integral
15 real :: start, finish, time
16 real(8) :: sigma
17
18 ! initialize the MPI environment
19 call MPI_INIT(ierr)
20 call MPI_COMM_SIZE(MPI_COMM_WORLD, ntasks, ierr)
21 call MPI_COMM_RANK(MPI_COMM_WORLD, rank, ierr)
22 allocate(status(MPI_STATUS_SIZE))
23
24 call CPU_TIME(start)
25
26 if (rank == 0) then
27     call SYSTEMCLOCK(clock)
28     call RANDOMSEED(size = n)
29     allocate(seed(n))

```

```

30      do i = 1, n
31          seed(i) = clock + 37 * i
32      end do
33      call RANDOMSEED(PUT = seed)
34      deallocate(seed)
35      do i = 1, ntasks - 1
36          call RANDOMNUMBER(z)
37          clock = clock + Int(z * 1000000)
38          call MPLSEND(clock, 1, MPLINTEGER, i, i, MPLCOMM_WORLD, ierr)
39      end do
40  else
41      call MPLRECV(clock, 1, MPLINTEGER, 0, rank, MPLCOMM_WORLD, status, ierr)
42      call RANDOMSEED(size = n)
43      allocate(seed(n))
44      do i = 1, n
45          seed(i) = clock + 37 * i
46      end do
47      call RANDOMSEED(PUT = seed)
48      deallocate(seed)
49  end if
50
51  do i = 1, n_tot
52      call RANDOMNUMBER(phi)
53      phi = phi * (phi_u - phi_l) + phi_l
54      call RANDOMNUMBER(rho)
55      rho = rho * (rho_u - rho_l) + rho_l
56      call RANDOMNUMBER(z)
57      z = z * (z_u - z_l) + z_l
58      call RANDOMNUMBER(f)
59      f = f * (f_u - f_l) + f_l
60      call func(phi, rho, z, f_real)
61      call inArea(phi, rho, z, H)
62      if (f < f_real) then
63          n_in = n_in + H
64      end if
65  end do
66  integral_local = (phi_u - phi_l) * (rho_u - rho_l) * (z_u - z_l) * (f_u - f_l) *
67      dble(n_in) / dble(n_tot)
68
69  call MPLREDUCE(integral_local, integral, 1, MPLREAL8, MPLSUM, 0, MPLCOMM_WORLD,
70      ierr)
71
72  if (rank == 0) then
73      integral = integral / dble(ntasks)
74      write(*, '(f10.5)') integral

```

```
73     end if
74
75     call CPU_TIME(finish)
76     call MPLREDUCE(finish - start, time, 1, MPIREAL, MPLSUM, 0, MPLCOMM_WORLD, ierr
77                  )
78     if (rank == 0) then
79         write(*, '("Time_consumed:_" f10.5) ') time
80     end if
81     call MPLREDUCE(((integral_local - dble(648) * pi / dble(5))**2)**2, sigma, 1,
82                  MPIREAL8, MPLSUM, 0, MPLCOMM_WORLD, ierr)
83     if (rank == 0) then
84         write(*, '("Error:_" f10.5) ') sqrt(sigma / dble(ntasks))
85     end if
86
87     call MPI_FINALIZE(ierr)
88 end program main
89
90 subroutine func(phi, rho, z, f)
91     ! the function to be integrated
92     implicit none
93     real(8), intent(in) :: phi, rho, z
94     real(8), intent(out) :: f
95
96     f = rho**3
97 end subroutine func
98
99 subroutine inArea(phi, rho, z, H)
100     ! judge whether the dot is in the integral area
101     real(8), intent(in) :: phi, rho, z
102     integer, intent(out) :: H
103
104     if ((z > -sqrt(9.d0 - rho**2)) .and. (z < sqrt(9.d0 - rho**2))) then
105         H = 1
106     else
107         H = 0
108     end if
109 end subroutine inArea
```