8. Random Number Generation
Pseudo-Random Numbers

- To simulate noise signals in the real world (true random numbers)
  e.g., background ambient noise in acoustics; turbulence in fluid flows
- To generate values for genetic testing of programs or subprograms

**Intrinsic pseudo-random number subroutine:**

```fortran
program intr_random_1
  implicit none
  integer :: n
  real :: arr(10)
  !
  call random_seed()
  call random_number(arr)
  !
  write(*,100) (arr(n),n=1,10)
end program
```

```fortran
program intr_random_2
  implicit none
  integer :: n
  real :: arr(10)
  !
  call random_seed()
  call random_number(arr)
  do n=1,10
    call random_number(arr(n))
  end do
  !
  write(*,100) (arr(n),n=1,10)
end program
```

- restart the random number generator
- return random number(s) from a uniform distribution in \([0, 1)\)
- `harvest` can be a scalar or an array

```
0.997560 0.566825 0.965915 0.747928 0.367391
0.480637 0.073754 0.005355 0.347081 0.342244
```
• The random numbers generate by `random_number` are uniformly distributed within \([0, 1)\).

To generate random numbers within \([-1, 1)\),
- rescale the numbers to \([0, 2)\),
- then shift the center to 0.

\[
\text{ran}=2.0*\text{ran}-1.0
\]

• Since the random numbers are uniform distributed within \([0, 1)\), the mean of the number should converge to 0.5.

The intrinsic random number generation subroutine can be tested by showing if the average of consecutive random numbers converges to 0.5 when the call increases.
program t_intr_random
! A test driver for the intrinsic random number generator subroutine.
! Average consecutive random numbers, the limit should be 0.5
implicit none
integer :: i, n, num
real :: avg, avg_0
real, allocatable :: arr(:)
!
call random_seed()
!
do i=1,7
    num=10**i
    allocate(arr(num))
    avg = 0.
    avg_0 = 0.
    do n=1,num
        call random_number(arr(n))
        avg = avg +arr(n)
        avg_0 = avg_0 +(2.*arr(n)-1.)
    end do
    avg = avg/real(num)
    avg_0 = avg_0/real(num)
    write(*,100) num, avg, avg_0
    deallocate(arr)
end do
!
100 format(i12,2f12.6)
A portable random number algorithm can be programmed based on the unpredictability of the modulo function:

\[ n_{i+1} = \text{MOD}(8121n_i + 28411, 134456) \]
\[ \text{ran}_i = \frac{n_i}{134456} \]

repeats after 134456 recursions

A random number generator returns a different and apparently random number (pseudo-random number) each time it is called. The numbers, in fact, are generated by a deterministic algorithm, and will eventually repeat!

FUNCTION random(iseed)
! When first call, iseed must be a large positive integer.
! iseed will be changed when exit and be used for next calling.
! The range of the generated random number is between 1 and -1
! implicit none
integer, intent(inout) :: iseed
real :: random
!
iseed = mod(8121*iseed+28411, 134456) ! 0 <= iseed < 134456
random = real(iseed)/134456.*2. -1. ! -1 < random < 1
!
end FUNCTION random

(See Chapter 7 in Numerical Recipes: The art of Scientific Programming)
PROGRAM t_random
! A test driver for the random number generator FUNCTION random
!
implicit none
integer :: i, n, num, iseed
real :: random, avg
!
!--Print out 10 random numbers
iseed=98765
  do n=1,10
    write(*,100) iseed, random(iseed)
  end do
100 format(i12,f12.6)
!
!--Average consecutive random numbers, the limit should be 0
  do i=1,7
    num = 10**i
    avg=0.
    do n=1,num
      avg = avg +random(iseed)
    end do
    avg = avg/real(num)
    write(*,100) num, avg
  end do
end PROGRAM
• **intrinsic random number generator:** `random_number`

```
<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>-0.021062</td>
</tr>
<tr>
<td>100</td>
<td>0.055202</td>
</tr>
<tr>
<td>1000</td>
<td>0.017216</td>
</tr>
<tr>
<td>10000</td>
<td>0.002165</td>
</tr>
<tr>
<td>100000</td>
<td>-0.001794</td>
</tr>
<tr>
<td>1000000</td>
<td>-0.000111</td>
</tr>
<tr>
<td>10000000</td>
<td>-0.000077</td>
</tr>
</tbody>
</table>
```

• **portable random number generator using modulo function:** `random`

```
<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.210262</td>
</tr>
<tr>
<td>100</td>
<td>-0.102290</td>
</tr>
<tr>
<td>1000</td>
<td>0.037673</td>
</tr>
<tr>
<td>10000</td>
<td>-0.010208</td>
</tr>
<tr>
<td>100000</td>
<td>-0.000220</td>
</tr>
<tr>
<td>1000000</td>
<td>-0.000079</td>
</tr>
<tr>
<td>10000000</td>
<td>-0.000005</td>
</tr>
</tbody>
</table>
```
Example:

A comparison test for three random number generators

• intrinsic function: \texttt{rand} (Fortran 77)
• intrinsic subroutine: \texttt{random\_number} (Fortran 2003 standard)
• portable subroutine: \texttt{random} (based on modulo function)
program t_comp_random
implicit none
integer, parameter :: nmax = 10000000
integer :: iseed = 43566, i
real, parameter :: error = 1.0e-06
real :: arr(nmax), random

!-- Part I: intrinsic function rand (Fortran 77)
call srand(iseed)
do i = 1, nmax
   arr(i) = 2.*rand()-1.
end do
write(*,*) 'rand:'
do i = 2, nmax
   if (abs(arr(1)-arr(i)) < error) then
      write(*,100) i
      exit
   end if
end do
do i = 3, nmax
   if (abs(arr(2)-arr(i)) < error) then
      write(*,101) i
      exit
   end if
end do
write(*,*),''

>Continued on next page...
call random_seed()
call random_number(arr)
arr = 2.*arr-1.
write(*,*) 'random_number:
do i = 2, nmax
    if (abs(arr(1)-arr(i)) < error) then
        write(*,100) i
        exit
    end if
end do
doi = 3, nmax
    if (abs(arr(2)-arr(i)) < error) then
        write(*,101) i
        exit
    end if
end do
write(*,*),''}
!-- Part III: portable algorithm: random
arr(1) = random(iseed)
do i = 2, nmax
    arr(i) = random(iseed)
end do
write(*,*) 'random:'
do i = 2, nmax
    if (abs(arr(1)-arr(i)) < error) then
        write(*,100) i
        exit
    end if
end do
do i = 3, nmax
    if (abs(arr(2)-arr(i)) < error) then
        write(*,101) i
        exit
    end if
end do
!
100 format('1st number is repeated at ',i8,' th number')
101 format('2nd number is repeated at ',i8,' th number')
!
end program
Random numbers repeat with different recurrent periods.

**rand:**
1st number is repeated at 1511326 th number
2nd number is repeated at 265037 th number

**random_number:**
1st number is repeated at 481954 th number
2nd number is repeated at 406187 th number

**random:**
1st number is repeated at 134457 th number
2nd number is repeated at 134458 th number

All random numbers repeat with a same recurrent period.