

Exercise n. II - part (a) (Pseudo)random numbers with uniform distribution

1. Intrinsic generators: uniformity and correlation (qualitative tests)

Use a *pseudorandom numbers intrinsic generator* to generate a sequence of random numbers (in this exercise don't worry about the seed).

For instance, in Fortran 90 `random_number()` is an *intrinsic* subroutine generating real random numbers in the range $[0,1[$.

The argument of the subroutine `random_number()` is `real`, has `intent out`, and can be a scalar or an array. See for instance `rantest_intrinsic.f90`, that shows different ways of calling the subroutine, generating one number at a time, or a sequence of the desired length. Note in the example the use of the dynamical allocation of memory (the instruction `allocatable`) for the array `x` and the use of `print` instruction with a specified format.

- (a) Produce a sequence of random numbers in $[0,1[$. Test the *uniformity* of the produced numbers making an histogram. Use for instance **gnuplot** with the command `w[ith] boxes`. Is the distribution *uniform*?

Hint: to do the histogram, divide the range into a given number of channels of width Δr , then calculate how many points fall in each channel, $r/\Delta r$:

```
integer, dimension(20) :: histog
:
histog = 0
do n = 1, ndata
  i = int(r(n)/delta_r) + 1
  histog(i) = histog(i) + 1
end do
```

- (b) We can test the presence of *correlation*. Consider the sequence of random numbers and plot the points (without connecting them with lines) corresponding to the pairs of consecutive numbers in the sequence:

$$(x_i, y_i) = (r_{2i-1}, r_{2i}) \quad i = 1, 2, 3, \dots$$

How many points (different pairs) would you expect? What do you see from the plot?

```

program rantest_intrinsic
!
! test program, call to intrinsic f90 random number generator
! generate random numbers in [0,1[ ; then,
! generate random integers between n_min and n_max.
!
implicit none
real :: rnd
real, dimension (:), allocatable :: x
integer :: L,i,n_min,n_max,ran_int

! generates ONE random number in [0,1[
call random_number(rnd)
print *, ' A real random number in [0,1[ is:',rnd

! generates L random numbers in [0,1[
print*, ' How many random numbers do you want to generate in [0,1[ ?'
print*, ' Insert the length of the sequence >'
read(*,*)L          ! length of sequence
do i = 1,L
call random_number(rnd)
print *,rnd
end do

! generates integer random numbers between n_min and n_max
print*, ' Generate ',L,' integer random numbers in [n_min,n_max[ ;'
print*, ' insert n_min, n_max >'
read(*,*),n_min,n_max
do i = 1,L
call random_number(rnd)
ran_int = (n_max - n_min + 1)*rnd + n_min
print *,ran_int
end do

! use array x to generate and store L random numbers with a unique call
print*, ' Generate other ',L,' real random numbers in [0,1[:'
allocate(x(L))
call random_number(x)
print*, x
deallocate(x)
end program rantest_intrinsic

```

2. Intrinsic generators: uniformity and correlation (quantitative tests)

Consider again a sequence generated by an intrinsic pseudorandom number generator.

(a) For a *uniformity* quantitative test, calculate the moment of order k :

$$\langle x^k \rangle^{calc} = \frac{1}{N} \sum_{i=1}^N x_i^k,$$

that should correspond to

$$\langle x^k \rangle^{th} = \int_0^1 dx x^k p_u(x) = \frac{1}{k+1}$$

where $p_u(x)$ is the uniform distribution in $[0,1]$. For a given k (fix for instance $k=1, 3, 7$), consider the deviation of the calculated momentum from the expected one: $\Delta_N(k) = |\langle x^k \rangle^{calc} - \langle x^k \rangle^{th}|$, and study its behaviour with N (N up to ~ 100.000). It should be $\sim 1/\sqrt{N}$. (*a log-log plot could be useful*)

(b) For a *correlation* quantitative test, for $k \neq 0$ calculate:

$$C(k)^{calc} = \frac{1}{N} \sum_{i=1}^N x_i x_{i+k}$$

that should correspond to

$$C^{th} = \int_0^1 dx \int_0^1 dy xy p_u(x)p_u(y) = \frac{1}{4}.$$

Consider the deviation of the calculated quantity from the expected one: $\Delta_N(k) = |C(k)^{calc} - 1/4|$ and study its behaviour with N (N up to ~ 100.000). It should be $\sim 1/\sqrt{N}$.

3. Intrinsic generators - seed [HERE: specifically for FORTRAN90]

The subroutine `random_seed` ([size] [put] [get]) initializes the sequence (it can be useful to control the initialization in case you want for instance to reproduce exactly a sequence of random numbers), but it can also return informations on the random numbers generator. It does not need necessarily an argument, and you cannot use more than one. The output variable `SIZE` is a scalar integer and gives the dimension `N` of the integer array (`SEED`). `SIZE` is compiler and machine dependent. The input variable `PUT` is the array of integers supplied by the user and that are used to initialize the sequence. The output variable `GET` (which is also an integer array) reads the instantaneous value of `SEED`. If no argument is given, the seed is initialized depending on the processor.

The code `rantest_intrinsic_with_seed.f90` (a bit more complicate than the previous one) calls also `random_seed`, in a way which is valid for any dimension of the seed.

The relevant part is:

```
call random_seed (sizer)
allocate(seed(sizer))
allocate(seed_old(sizer))
...
call random_seed (put = seed )           ! Initialized by the user
call random_seed (get = seed_old )      ! Gives the present values
```

Here there is another example (`rantestbis_intrinsic.f90`), where you can see three different sequences (with changes of the seed):

```
program rantestbis_intrinsic
! test program, call to intrinsic random number f90 generator
! illustrate the use of "put" and "get" in call random_seed
implicit none
integer, dimension(:), allocatable :: seed, seed_old
integer :: L,i,sizer,n_min,n_max,ran_int
integer :: i, sizer
real, dimension(3) :: harvest

call random_seed(sizer)
allocate(seed(sizer))
allocate(seed_old(sizer))
print *, 'Here the seed has ',sizer,' components; insert them (or print "/" ) >'
read(*,*)seed

call random_seed(put=seed)
call random_seed(get=seed_old)
```

```

print*, "Old starting value: ",seed_old

call random_number(harvest)
print*,"3 random numbers: ",harvest

do i=1,3
  call random_seed(get=seed_old)
  print*,"Present values of seed: ",seed_old
  call random_number(harvest)
  print*,"Other 3 random numbers: ",harvest
  call random_number(harvest)
  print*,"and other 3 random numbers: ",harvest
end do

deallocate(x)
deallocate(seed)
deallocate(seed_old)

end program rantestbis_intrinsic

```

Play with the use of `random_seed` and its arguments. It is instructive to run the same code on different compilers/machines (or compare results with classmates). What about the *size* of the seed, for instance?

4. Intrinsic generators - initialisation [HERE: specifically for FORTRAN90]

Check whether the seed is changed automatically or not using `random_seed()` subroutine in your computer (run the code for random number generation more than once). In general, you should see that the initialisation is "processor dependent".

You can force a change using the system clock, like in the following subroutine. Do experiments!

```

SUBROUTINE init_random_seed
INTEGER :: i, nseed, clock
INTEGER, DIMENSION(:), ALLOCATABLE :: seed

CALL RANDOM_SEED(size = nseed)
ALLOCATE(seed(nseed))
CALL SYSTEM_CLOCK(clock)

seed = clock/2 + 37 * (/ (i - 1, i = 1, n) /)
CALL RANDOM_SEED(PUT = seed)

DEALLOCATE(seed)
END SUBROUTINE

```