



Science and  
Technology  
Facilities Council



ÉCOLE  
POLYTECHNIQUE  
MONTRÉAL



# GALAHAD

# RAND

USER DOCUMENTATION

GALAHAD Optimization Library version 4.2

## 1 SUMMARY

GALAHAD\_RAND is a suite of Fortran procedures for generating **uniformly distributed pseudo-random numbers**. Random reals are generated in the range  $0 < \xi < 1$  or the range  $-1 < \eta < 1$  and random integers in the range  $1 \leq k \leq N$  where  $N$  is specified by the user.

A multiplicative congruent method is used where a 31 bit generator word  $g$  is maintained. On each call to a procedure of the package,  $g_{n+1}$  is updated to  $7^5 g_n \bmod (2^{31} - 1)$ ; the initial value of  $g$  is  $2^{16} - 1$ . Depending upon the type of random number required the following are computed  $\xi = g_{n+1} / (2^{31} - 1)$ ;  $\eta = 2\xi - 1$  or  $k = \text{integer part } \xi N + 1$ .

The package also provides the facility for saving the current value of the generator word and for restarting with any specified value.

**ATTRIBUTES — Versions:** GALAHAD\_RAND\_single, GALAHAD\_RAND\_double, **Uses:** None. **Date:** March 2001. **Origin:** N. I. M. Gould and J. K. Reid, Rutherford Appleton Laboratory. **Language:** Fortran 95 + TR 15581 or Fortran 2003.

## 2 HOW TO USE THE PACKAGE

Access to the package requires a USE statement such as

*Single precision version*

```
USE GALAHAD_RAND_single
```

*Double precision version*

```
USE GALAHAD_RAND_double
```

If it is required to use both modules at the same time, the derived type RAND\_seed (Section 2.1) and the sub-routines RAND\_random\_real, RAND\_random\_integer, RAND\_get\_seed, and RAND\_set\_seed (Section 2.2) must be re-named on one of the USE statements. Their seeds will be independent.

### 2.1 The derived data types

The user must provide a variable of derived type RAND\_seed to hold the current seed value which must be passed to all calls of RAND. The seed value component is private and can only be set and retrieved through the RAND\_set\_seed and RAND\_get\_seed entries.

### 2.2 Argument lists and calling sequences

There are five procedures for user calls. The initialization entry must be called before any call to the RAND\_random\_real, RAND\_random\_integer and RAND\_get\_seed entries.

#### 2.2.1 Subroutine to initialize the generator word

This entry must be called first to initialize the generator word.

```
CALL RAND_initialize( seed )
```

seed is a scalar INTENT (OUT) argument of derived type RAND\_seed that holds the seed value.

**All use is subject to the conditions of a BSD-3-Clause License.**

**See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.**

### 2.2.2 Subroutine to obtain random real values

A random real value or values may be obtained as follows:

```
CALL RAND_random_real( seed, positive, random_real )
```

`seed` is a scalar `INTENT(INOUT)` argument of derived type `RAND_seed` that holds the seed value.

`positive` is a scalar `INTENT(IN)` argument of type default `LOGICAL`. If `positive` is `.TRUE.`, the generated random number is a real value in the range  $0 < \xi < 1$ , while if `positive` is `.FALSE.`, the generated random number is a real value in the range  $-1 < \eta < 1$ .

`random_real` is a scalar or rank 1 or 2 array `INTENT(OUT)` argument of type `REAL` (double precision in `GALAHAD_RAND_double`). It is set to the required random number(s).

### 2.2.3 Subroutine to obtain random integer values

A random integer value or values may be obtained as follows:

```
CALL RAND_random_integer( seed, n, random_integer )
```

`seed` is a scalar `INTENT(INOUT)` argument of derived type `RAND_seed` that holds the seed value.

`n` is a scalar `INTENT(IN)` argument of type default `INTEGER`. It must be set by the user to specify the upper bound for the range  $1 \leq k \leq n$  within which the generated random number(s)  $k$  is/are required to lie. **Restriction:** `n` must be positive.

`random_integer` is a scalar or rank 1 or 2 array `INTENT(OUT)` argument of type default `INTEGER`. It is set to the required random integer  $k$  or an array of such integers.

### 2.2.4 Subroutine to obtain the current generator word

The current generator word may be obtained as follows:

```
CALL RAND_get_seed( seed, value )
```

`seed` is a scalar `INTENT(IN)` argument of derived type `RAND_seed` that must be provided to hold the seed value.

`value` is a scalar `INTENT(OUT)` argument of type default `INTEGER`. It is set to the current value of the generator word  $g$ .

### 2.2.5 Subroutine to reset the current value of the generator word

The current value of the generator word may be reset as follows:

```
CALL RAND_set_seed( seed, value )
```

`seed` is a scalar `INTENT(OUT)` argument of derived type `RAND_seed` that holds the seed value.

`value` is a scalar `INTENT(IN)` argument of type default `INTEGER` that must be set by the user to the required value of the generator word. It is recommended that the value should have been obtained by a previous call of `RAND_get_seed`. It should have a value in the range  $0 < \text{value} \leq P$ , where  $P = 2^{31} - 1 = 2147483647$ . If it is outside this range, the value  $\text{value} \bmod (2^{31} - 1)$  is used.

---

**All use is subject to the conditions of a BSD-3-Clause License.**

**See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.**

### 3 GENERAL INFORMATION

**Use of common:** None.

**Workspace:** None.

**Other routines called directly:** None.

**Other modules used directly:** None.

**Input/output:** None.

**Restrictions:**  $n > 0$ .

**Portability:** ISO Fortran 95 + TR 15581 or Fortran 2003. The package is thread-safe.

### 4 METHOD

The code is based on that of L.Schrage, “A More Portable Fortran Random Number Generator”, TOMS, **5**(2) June 1979. The method employed is a multiplicative congruential method. The generator word  $g$  is held as an integer and is updated on each call as follows

$$g_{n+1} = 7^5 g_n \bmod (2^{31} - 1)$$

The result returned from `RAND_random_real`, for a non-negative argument, is  $\xi$ , where

$$\xi = g_{n+1} / (2^{31} - 1)$$

and for a negative argument is

$$2\xi - 1.$$

The value of  $k$  returned by `RAND_random_integer` is

$$\text{integer part } \xi N + 1.$$

Arrays of random reals and integers are formed by calling the above sequentially in Fortran column order.

### 5 EXAMPLE OF USE

Suppose we wish to generate two random real numbers lying between plus and minus one, reset the generator word to its original value, and then find two positive random integers with values no larger than one hundred. Then we might use the following piece of code.

```
! THIS VERSION: GALAHAD 2.6 - 03/07/2014 AT 13:00 GMT.
PROGRAM GALAHAD_RAND_spec
  USE GALAHAD_RAND_double
  IMPLICIT NONE
  TYPE (RAND_seed) seed
  INTEGER :: random_integer, value
  REAL ( kind = KIND( 1.0D+0 ) ) :: random_real
! Initialize the generator word
  CALL RAND_initialize( seed ) ! Get the current generator word
  CALL RAND_get_seed( seed, value )
  WRITE( 6, "( ' generator word = ', I0 )" ) value
! Generate a random real in [-1, 1]
  CALL RAND_random_real( seed, .FALSE., random_real )
```

---

**All use is subject to the conditions of a BSD-3-Clause License.**

**See <http://galahad.rl.ac.uk/galahad-www/cou.html> for full details.**



```

WRITE( 6, "( ' random real = ', F5.2 )" ) random_real
! Generate another random real
CALL RAND_random_real( seed, .FALSE., random_real )
WRITE( 6, "( ' second random real = ', F5.2 )" ) random_real
! Restore the generator word
CALL RAND_set_seed( seed, value )
! Generate a random integer in [1, 100]
CALL RAND_random_integer( seed, 100, random_integer )
WRITE( 6, "( ' random integer in [1,100] = ', I0 )" ) random_integer
! Generate another random integer
CALL RAND_random_integer( seed, 100, random_integer )
WRITE( 6, "( ' second random integer in [1,100] = ', I0 )" ) random_integer
END PROGRAM GALAHAD_RAND_spec

```

This produces the following output:

```

generator word = 65535
random real = 0.03
second random real = -0.34
random integer in [1,100] = 52
second random integer in [1,100] = 33

```