JMASM1: RANGEN 2.0 (Fortran 90/95)

Gail F. Fahoome
Wayne State University

Follow this and additional works at: http://digitalcommons.wayne.edu/jmasm

Part of the Applied Statistics Commons, Social and Behavioral Sciences Commons, and the Statistical Theory Commons

Recommended Citation
DOI: 10.22237/jmasm/1020255960
Available at: http://digitalcommons.wayne.edu/jmasm/vol1/iss1/25

This Algorithms and Code is brought to you for free and open access by the Open Access Journals at DigitalCommons@WayneState. It has been accepted for inclusion in Journal of Modern Applied Statistical Methods by an authorized editor of DigitalCommons@WayneState.
Rangen 2.0 is Fortran 90 module of subroutines used to generate uniform and nonuniform pseudo-random deviates. It includes unil, an uniform pseudo-random number generator, and non-uniform generators based on unil. The subroutines in Rangen 2.0 were written using Essential Lahey Fortran 90, a proper subset of Fortran 90. It includes both source code for the subroutines and a short description of each subroutine, its purpose, and the arguments including data type and usage.

Introduction

Rangen is a collection of subroutines used to generate uniform and nonuniform pseudo-random deviates. Rangen 1.0, the original version, was written in Fortran 77 (IBM RT PC FORTRAN 77, Version 1.01) by R. Clifford Blair and published by IBM in 1987. It was developed for the IBM AIX Unix platform. Shlomo S. Sawilowsky produced Rangen 1.1, a minor translation of Rangen 1.0, for the PC environment, and added additional subroutines which are not included here. Rangen 2.0 is a translation of selected Rangen 1.1 subroutines from Fortran 77 to Fortran 90 using Essential Lahey Fortran 90 (ELF) (1998, Version 4.0). ELF is a proper subset of Fortran 90. The selected subroutines have been incorporated into a module.

The original version of Rangen included two uniform pseudo-random number generators, called Unil and Uni2. Blair noted that both had been extensively tested and “appear to be excellent generators” (1987, p. I-1). There were some differences between the two. For example, Unil is faster than Uni2, and thus may produce slightly less random deviates. However, “In most cases either generator will prove to be quite satisfactory” (Blair, p. I-1). Rangen 2.0 includes Uni1 and the nonuniform pseudo-random number generators based on Uni1. The names of the subroutines have retained the designation ‘1’ so that Uni2 and the nonuniform pseudo-random number generators based on Uni2 can be added at a later date.

The material is presented in two sections. The first section contains the source code for the module Rangen 2.0. The second section contains a short description of each subroutine, its purpose, and the arguments including data type and usage. An example of a program segment is included for each subroutine, including the type declarations, initialization of variables, and the subroutine call. The output of the example is given so that the user may check output for accuracy.

The references for the algorithms used in the subroutines written by Blair are included in the description section of Rangen 2.0. Subroutine Normbl, and Ppnd2, a subroutine called by Normbl, are adaptations of code written by Beasley and Springer (1985). The citation and original permission to reproduce published in Rangen 1.0 are contained in the description of Normbl.

Source Code for Rangen 2.0

module rangen
    implicit none
    public:: cayl, chisq1, erll, exp1, fl, lapl, lnor1, normbl, tl, unil, ppnd2
contains
    subroutine cayl(dseed, inp, x)
        integer:: i
        integer, intent(in):: inp
        real, intent(out):: x(inp)
        real:: u(2), v1, v2
        real (kind=8), intent(in out):: dseed
        do i = 1, inp
            call unil(dseed, 2, u)
            v1 = 2 * u(1)-1
            v2 = 2 * u(2)-1
            if (v1**2+v2**2 .gt. 1) go to 10
            x(i) = v1/v2
        end do
        return
    end subroutine cayl
    subroutine chisq1(dseed, inp, x)
        integer:: i
        integer, intent(in):: inp
        real, intent(out):: x(inp)
        real:: u(2), v1, v2
        real (kind=8), intent(in out):: dseed
        do i = 1, inp
            call uni1(dseed, 2, u)
            v1 = 2*u(1)-1
            v2 = 2*u(2)-1
            if (v1**2+v2**2 .gt. 1) go to 10
            x(i) = v1/v2
        end do
        return
    end subroutine chisq1
end module
integer, intent(in):: inp, idf
integer:: i, idf1
real, intent(inout):: x(inp), y(inp)
real (kind=8), intent(inout):: dseed
if (idf == 1) go to 30
if (mod(idf, 2) /= 0) go to 40
idfl = idf/2
call erll(dseed, inp, idfl, 2.0, y, x)
return
30 call normbl(dseed, inp, x)
do i = 1, inp
x(i) = x(i)**2
end do
return
40 idfl = (idf-1)/2
call erll(dseed, inp, idfl, 2.0, y, x)
call normb1(dseed, inp, y)
do i = 1, inp
x(i) = x(i)+y(i)**2
end do
return
end subroutine chisq1
!
subroutine erll(dseed, inp, ia, b, y, x)
integer, intent(in):: inp
integer, intent(in):: ia
integer:: i, j
real, intent(in):: b
real, intent(inout):: y(inp)
real, intent(inout):: x(inp)
real:: py
real (kind=8), intent(inout):: dseed
if (ia > 50) go to 30
do i = 1, inp
py = y(i)
do j = 2, ia
py = py*y(j)
end do
x(i) = -b*log(py)
end do
return
30 do i = 1, inp
py = y(i)
do j = 2, 50
py = py*y(j)
end do
x(i) = -b*log(py)
py = 0.0
do j = 51, ia
py = py+log(y(j))
end do
x(i) = -b*py+x(i)
end do
return
end subroutine exp1
!
subroutine f1(dseed, inp, idfn, idfd, y1, y2, x)
integer, intent(in):: inp, idfn, idfd
integer:: i
real, intent(out):: x(inp)
real, intent(inout):: y1(inp), y2(inp)
real (kind=8), intent(inout):: dseed
call chisq1(dseed, inp, idfn, y2)
call exp1(dseed, inp, 1.0, x)
do i = 1, inp
if (y(i)>= .5) x(i) = -x(i)
end do
return
end subroutine f1
!
subroutine lap1(dseed, inp, y, x)
integer, intent(in):: inp
integer:: i
real, intent(out):: x(inp)
real, intent(inout):: y(inp)
call unil(dseed, inp, y)
call expl(dseed, inp, 1.0, x)
do i = l, inp
if (y(i)>= .5) x(i) = -x(i)
end do
return
end subroutine lap1
!
subroutine lnor1(dseed, inp, am, sd, x)
integer, intent(in):: inp
integer:: i
real, intent(in):: am, sd
real, intent(out):: x(inp)
real (kind=8), intent(inout):: dseed
call normb1(dseed, inp, x)
do i = 1, inp
x(i) = exp(sd*x(i)+am)
end do
return
end subroutine lnor1
!
subroutine normbl(dseedl, inp1, x1)
integer, intent(in):: inp1
integer:: i, ifault
real, intent(out):: x1(inp1)
real(kind=8), intent(in out):: dseedl
real:: xtemp
call unil(dseedl, inp1, x1)
do i = 1, inp1
call ppnd2(x1(i), xtemp, ifault)
x1(i) = xtemp
end do
return
end subroutine normbl
!
subroutine t1(dseed, inp, idf, y, x)
integer, intent(in):: inp, idf
integer:: i
real, intent(out):: x(inp)
real, intent(in out):: y(inp)
real(kind=8), intent(in out):: dseed
call chisql(dseed, inp, idf, y, x)
call normbl(dseed, inp, y)
do i = 1, inp
x(i) = y(i)/sqrt((x(i)/real(idf)))
end do
return
end subroutine t1
!
subroutine unil(dseed, inp, x)
real(kind=8), intent(in out):: dseed
integer, intent(in):: inp
integer:: i
real, intent(out):: x(inp)
do i = 1, inp
dseed = modulo(16807._8*dseed, 2147483647._8)
x(i) = dseed/2147483648._8
end do
return
end subroutine unil
!
subroutine ppnd2(p, ppndt, ifault)
real:: zero, split, half, one, a0, a1, a2, a3, b1, b2, b3, b4, &
c0, c1, c2, c3, d1, d2, q, r
real,intent(in):: p
real, intent(out):: ppndt
integer, intent(out):: ifault
zero = 0.0e0
half = 0.5e0
one = 1.0e0
split = 0.42e0
a0 = 2.50662823884e0
a1 = -18.61500062529e0
a2 = -41.39119773534e0
a3 = -25.44106049637e0
b1 = -8.47351093090e0
b2 = 23.08336743743e0
b3 = -21.06224101826e0
b4 = 3.13082909833e0
c0 = -2.78718931138e0
c1 = -2.29796479134e0
c2 = 4.85014127135e0
c3 = 2.32121276858e0
d1 = 3.54388924762e0
d2 = 1.63706781897e0
ifault=0
q = p- half
if (abs(q) > split) go to 1
r = q*q
ppndt = q*(((a3*r+a2)*r+a1)*r+a0)/((((b4*r+b3)*r+b2)*r+b1)*r+one)
return
1 r = p
if (q > zero) r = one _ p
if (r <= zero) go to 2
r = sqrt(-log(r))
ppndt = (((c3*r+c2)*r+c1)*r+c0)/(((d2*r+d1)*r+one))
if (q < zero) ppndt = _ppndt
return
2 ifault = 1
ppndt = zero
return
end subroutine ppnd2
!
end module rangen
Description of Rangen 2.0 Subroutines

CAY1

Cay1 generates deviates from a Cauchy distribution.

Arguments:
- dseed - Input/output. Dseed must be an integer of type real (kind=8) in the exclusive range 1_8 to 2147483647_8. Cay1 returns a new dseed.
- inp - Input: The number of deviates to be returned.
- x - Output: A vector of length inp containing the deviates.

Call:
```
call cay1(dseed, inp, x)
```

Example: Cay1 is used to generate 100 deviates.
```
real:: x(100)
real(kind=8):: dseed
dseed = 12346
call cay1(dseed, 100, x)
```

Output:
```
dseed = 944541922
x(1) = -37.1592
x(100) = 5.59855
```

CHISQ1

Chisq1 generates deviates from a chi-squared distribution with user-provided degrees of freedom.

Arguments:
- dseed - Input/output. Dseed must be an integer of type real (kind=8) in the exclusive range 1_8 to 2147483647_8. Chisq1 returns a new dseed.
- inp - Input: The number of deviates (integer) to be returned.
- idf - Input: The value (integer) of degrees of freedom.
- y - Work vector (real) of length inp.
- x - Output: A real vector of length inp containing the deviates.

Call:
```
call chisq1(dseed, inp, idf, y, x)
```

Example: Chisq1 is used to generate 100 deviates.
```
integer:: idf
integer, parameter:: inp=100
real:: x(inp), y(inp)
real(kind=8):: dseed
dseed = 12346
idf = 3
call chisq1(dseed, inp, idf, y, x)
```

Output:
```
dseed = 1533170485
x(1) = 4.69289
x(100) = 1.86385
```
ERL1
ERL1 generates deviates from a Erlang distribution with user-provided degrees of freedom.

Arguments:
- **dseed** - Input/output. Dseed must be an integer of type real (kind=8) in the exclusive range 1.8 to 2147483647.8. ERL1 returns a new dseed.
- **inp** - Input: The number of deviates (integer) to be returned.
- **ia** - Input: Shape parameter (integer) which must be greater than or equal to one.
- **b** - Input: Scale parameter (real) which must be greater than zero.
- **y** - Work vector of length inp.
- **x** - Output: A real vector of length inp containing the deviates.

Call:
```
call erl1(dseed, inp, ia, b, y, x)
```

Example: ERL1 is used to generate 100 deviates.
```
integer:: ia
integer, parameter:: inp=100
real:: x(inp), y(inp), b
real (kind=8):: dseed
dseed = 12346
ia = 2
b = 3.5
call erl1(dseed, inp, ia, b, y, x)
```

Output:
```
dseed = 1533170485
x(1) = 8.30200
x(100) = 1.27826
```

EXP1
EXP1 generates deviates from an exponential distribution with user-provided mean and standard deviation.

Arguments:
- **dseed** - Input/output: Dseed must be an integer of type real (kind=8) in the exclusive range 1.8 to 2147483647.8. EXP1 returns a new dseed.
- **inp** - Input: The number of deviates (integer) to be returned.
- **sm** - Input: Mean and standard deviation (real) of the sampled population.
- **x** - Output: A real vector of length inp containing the deviates.

Call:
```
call exp1(dseed, inp, sm, x)
```

Example: EXP1 is used to generate 100 deviates.
```
integer, parameter:: inp=100
real:: x(inp), sm
real (kind=8):: dseed
dseed = 12346
sm = 1.0
call exp1(dseed, inp, sm, x)
```

Output:
```
dseed = 991974008
x(1) = 2.33692
x(100) = 0.772355
```
FI generates deviates from a F distribution with user-provided degrees of freedom.

Arguments:  
- dseed: Input/output. Dseed must be an integer of type real (kind=8) in the exclusive range 1.8 to 2147483647.8. FI returns a new dseed.
- inp: Input: The number of deviates (integer) to be returned.
- idfn: Input: The value (integer) of degrees of freedom for the numerator of the f variable.
- idfd: Input: The value (integer) of degrees of freedom for the denominator of the f variable.
- y1: Work vector (real) of length inp.
- y2: Work vector

Call:  
call f1(dseed, inp, idfn, idfd, y1, y2, x)

Example:  
integer:: idfn, idfd  
integer, parameter:: inp=100  
real:: x(inp), y1(inp), y2(inp)  
real (kind=8):: dseed  
dseed = 12346  
idfn = 3  
idfd = 5  
call f1(dseed, inp, idfn, idfd, y1, y2, x)

Output:  
dseed = 580303867  
x(1) = 1.39239  
x(100) = 1.30237

LAP1 generates deviates from a Laplace (double exponential) distribution, using an exponential distribution with mean and standard deviation 1.0.

Arguments:  
- dseed: Input/output. Dseed must be an integer of type real (kind=8) in the exclusive range 1.8 to 2147483647.8. Lapl returns a new dseed.
- inp: Input: The number of deviates (integer) to be returned.
- y: Work vector (real) of length inp.
- x: Output: A real vector of length inp containing the deviates.

Call:  
call lap1(dseed, inp, y, x)

Example:  
Lapl is used to generate 100 deviates.  
integer, parameter:: inp=100  
real:: x(inp), y(inp)  
real (kind=8):: dseed  
dseed = 12346  
call lap1(dseed, inp, y, x)

Output:  
dseed = 1533170485  
x(1) = 0.588999  
x(100) = 0.336959
Lnor1 generates deviates from a lognormal distribution with user-provided parameters.

Arguments:  
dseed - Input/output: Dseed must be an integer of type real (kind=8) in the exclusive range 1.8 to 2147483647.8. Lnor1 returns a new dseed.
inp - Input: The number of deviates (integer) to be returned.
am - Input: The mean of y (real) for y = ln(x).
sd - Input: The standard deviation (real) of y for y = ln(x).
x - Output: A real vector of length inp containing the deviates.

Call:  
call lnor1(dseed, inp, am, sd, x)

Example:  
Lnor1 is used to generate 100 deviates.
integer, parameter:: inp=100
real:: x(inp), am, sd
real (kind=8):: dseed

dseed = 12346
am = 0.0
sd = sqrt(2.0)
call lnor1(dseed, inp, am, sd, x)

Output:  
dseed = 991974008
x(1) = 0.158828
x(100) = 0.873557

Normbl generates deviates from the standard normal distribution.

Arguments:  
dseed - Input/output: Dseed must be an integer of type real (kind=8) in the exclusive range 1.8 to 2147483647.8. Normbl returns a new dseed.
inp - Input: The number of deviates (integer) to be returned.
x - Output: A real vector of length inp containing the deviates.

Call:  
call normbl(dseed, inp, x)

Example:  
Normbl is used to generate 100 deviates.
integer, parameter:: inp=100
real:: x(inp)
real (kind=8):: dseed

dseed = 12346
call normbl(dseed, inp, x)

Output:  
dseed = 991974008
x(1) = -1.30103
x(100) = -0.095588

T1 generates deviates from an exponential distribution with user-provided degrees of freedom.

Arguments:
- dseed - Input/output: Dseed must be an integer of type real (kind=8) in the exclusive range 1.8 to 2147483647.8. T1 returns a new dseed.
- inp - Input: The number of deviates (integer) to be returned.
- idf - Input: The degrees of freedom (integer) for the t variable.
- y - Work vector (real) of length inp.
- x - Output: A real vector of length inp containing the deviates.

Call: call t1(dseed, inp, idf, y, x)

Example: T1 is used to generate 100 deviates.

```
integer:: idf
integer, parameter:: inp=100
real:: x(inp), y(inp)
real (kind=8):: dseed

dseed = 12346
idf = 3
call t1(dseed, inp, idf, y, x)
```

Output:
- dseed = 489858532
- x(1) = -0.800539
- x(100) = -0.945289

UNI1 generates deviates from the uniform distribution.

Arguments:
- dseed - Input/output: Dseed must be an integer of type real (kind=8) in the exclusive range 1.8 to 2147483647.8. Unil returns a new dseed.
- inp - Input: The number of deviates (integer) to be returned.
- x - Output: A real vector of length inp containing the deviates.

Call: call uni1(dseed, inp, x)

Example: Uni1 is used to generate 100 deviates.

```
integer, parameter:: inp=100
real:: x(inp)
real (kind=8):: dseed

dseed = 12346
call uni1(dseed, inp, x)
```

Output:
- dseed = 991974008
- x(1) = 0.0966244
- x(100) = 0.461924
References


