

# other random distributions

October 26, 1999  
21:13

## Contents

<b>1</b>	<b>Other random distributions</b>	<b>1</b>
<b>2</b>	<b>External interface</b>	<b>1</b>
<b>3</b>	<b>Normal random numbers</b>	<b>2</b>
<b>4</b>	<b>Isotropic distribution</b>	<b>4</b>
<b>5</b>	<b>Cosine distribution</b>	<b>6</b>
<b>6</b>	<b>INDEX</b>	<b>8</b>

## 1 Other random distributions

```
$Id: randother.web,v 1.5 1999/08/09 15:18:40 karney Exp $
```

```
"randother.f" 1 ≡
@m FILE 'randother.web'
```

## 2 External interface

Three distributions are provided: normal deviates, uniform points on a sphere, and cosine-distributed points on a half sphere. Here the state of the random number generator (which should already have been initialized via *random\_init*) is defined by *rn\_args(tag)*. Non-FWEB users will need to replace this with *ran\_index*, *ran\_array* (suitably declared).

**call** *random\_gauss*(*y*, *n*, *rn\_args(tag)*) returns *n* normal deviates in the array *y* with zero mean and unit variance.

**call** *random\_isodist*(*v*, *n*, *rn\_args(tag)*) returns *n* points uniformly distributed on a sphere in the array *v*<sub>3, *n*</sub>.

**call** *random\_cosdist*(*v*, *n*, *rn\_args(tag)*) returns a *n* points cosine-distributed on a sphere in the array *v*<sub>3, *n*</sub>, with the third *z* axis being the normal.

These routines all return 64-bit floating results. There are also single precision versions obtained by prefixing the routine names with an *s*.

```
"randother.f" 2 ≡
< Functions and Subroutines 3 >
```

### 3 Normal random numbers

Generate normal random numbers with zero mean and unit variance. This is the Box-Muller method [?] described in Knuth [?], p. 117–118. However, rather than use the rejection method to calculate  $\cos \Theta$  and  $\sin \Theta$ , we calculate them directly, to allow vectorization.

Because of the way the uniform random numbers are generated, this routine will never attempt to take  $\log 0$ . In addition, 0 will never be one of the normal random numbers generated. (These observations follow because *random* never returns 0,  $\frac{1}{4}$ ,  $\frac{1}{2}$ ,  $\frac{3}{4}$ , or 1.)

If  $n$  is odd, then one random number is essentially thrown away. This means, for example, that the state of the random number generator will be different after two calls to **call** *random\_gauss*( $y$ , 1, *rn\_dummy*( $x$ )) compared to a single call to **call** *random\_gauss*( $y$ , 2, *rn\_dummy*( $x$ )). (The first consumes 4 random numbers; the second consumes 2.)

⟨Functions and Subroutines 3⟩ ≡

```

subroutine random_gauss( $y$ ,  $n$ , rn_dummy( $x$ ))
  implicit_none_f77
  implicit_none_f90
  rn_decl( $x$ ) // RNG State
  integer  $n$  // Input
  real  $y_{0:n-1}$  // Output
  integer  $i$  // Local
  real  $pi$ ,  $theta$ ,  $z$ 
  real random // External
  external random_array, random
  data  $pi/const(3.14159265358979323846264338328)/$ 
  @#if -HIPREC
    single_precision  $ys_{0:n-1}$  // Output
    single_precision  $spi$ ,  $stheta$ ,  $sz$  // Local
    single_precision srandom // External
    external srandom_array, srandom
    data  $spi/3.14159265358979323846264338328/$ 
  @#endif
  @#if HIPREC
    entry srandom_gauss( $y$ ,  $n$ , rn_dummy( $x$ ))
  @#endif
  if ( $n \leq 0$ )
    return
  call random_array( $y$ ,  $n$ , rn_args( $x$ ))
  do  $i = 0$ ,  $\text{int}(n/2) * 2 - 1$ , 2
     $theta = pi * (two * y_i - one)$  // uniformly distributed in  $(-\pi, \pi)$ 
     $z = \text{sqrt}(-two * \log(y_{i+1}))$  //  $\sqrt{-2 \log S}$ 
     $y_i = z * \cos(theta)$ 
     $y_{i+1} = z * \sin(theta)$ 
  end do
  if ( $\text{mod}(n, 2) \equiv 0$ )
    return
     $theta = pi * (two * y_{n-1} - one)$  // uniformly distributed in  $(-\pi, \pi)$ 

```

```

z = sqrt(-two * random(rn_args(x))) //  $\sqrt{-2 \log S}$ 
yn-1 = z * cos(theta)

return

@if -HIPREC

entry srandom_gauss(ys, n, rn_dummy(x))

if (n ≤ 0)
  return

call srandom_array(ys, n, rn_args(x))
do i = 0, int(n / 2) * 2 - 1, 2
  stheta = spi * (2.0 * ysi - 1.0) // uniformly distributed in (-π, π)
  sz = sqrt(-2.0 * log(ysi+1)) //  $\sqrt{-2 \log S}$ 
  ysi = sz * cos(stheta)
  ysi+1 = sz * sin(stheta)
end do

if (mod(n, 2) ≡ 0)
  return
stheta = spi * (2.0 * ysn-1 - 1.0) // uniformly distributed in (-π, π)
sz = sqrt(-2.0 * srandom(rn_args(x))) //  $\sqrt{-2 \log S}$ 
ysn-1 = sz * cos(stheta)

return
@endif
end

```

See also sections 4 and 5.

This code is used in section 2.

## 4 Isotropic distribution

Generate random points isotropically on a three-dimensional unit sphere. Each of the coordinates is uniformly distributed  $[-1, 1]$ . We use this to obtain the  $z$  coordinate. Then  $x$  and  $y$  are uniformly distributed on a circle of radius  $\sqrt{1 - z^2}$ . See Carter and Cashwell [?], p. 7. There are rejection techniques to calculate these random numbers, but once again the additional complexity will prevent vectorization, so these techniques are unlikely to be faster.

Because of the way the uniform random numbers are calculated, this routine will never return 0 for any of the components of the position. (However, it *may* return 1 for  $x$  or  $y$  components.)

This and *random\_cosdist* dubiously treats its array output argument,  $v$  as a one-dimensional array. (This would normally be dimensioned as  $v_3, n$  in the calling program.) By doing this we can easily use  $v$  as a place to hold the random numbers returned by *random\_array*. Note that we fill up the last  $\frac{2}{3}$  of  $v$  in the call to *random\_array*, and then proceed to fill up  $v$  with the required random vectors starting at the beginning, ensuring that we don't overwrite any random numbers before they are needed.

(Functions and Subroutines 3) +≡

```

subroutine random_isodist(v, n, rn_dummy(x))
  implicit_none_f77
  implicit_none_f90
  rn_decl(x) // RNG State
  integer n // Input
  real v0:3*n-1 // Output
  integer i // Local
  real pi, costheta, phi
  external random_array // External
  data pi / const(3.14159265358979323846264338328) /
  @if -HIPREC
    single_precision vs0:3*n-1 // Output
    single_precision spi, scostheta, sphi // Local
    external srandom_array
    data spi / 3.14159265358979323846264338328 /
  @endif
  @if HIPREC
    entry srandom_isodist(v, n, rn_dummy(x))
  @endif
  if (n ≤ 0)
    return
  call random_array(vn, 2 * n, rn_args(x))
  do i = 0, n - 1
    costheta = two * vn+2*i - one // uniformly distributed in (-1, 1)
    phi = pi * (two * vn+2*i+1 - one) // uniformly distributed in (-π, π)
    v3*i = cos(phi) * sqrt(one - costheta2)
    v3*i+1 = sin(phi) * sqrt(one - costheta2)
    v3*i+2 = costheta
  end do
  return
  @if -HIPREC
    entry srandom_isodist(vs, n, rn_dummy(x))

```

```
if (n ≤ 0)
  return
call srandom_array(vs_n, 2 * n, rn_args(x))
do i = 0, n - 1
  scostheta = 2.0 * vs_{n+2*i} - 1.0 // uniformly distributed in (-1, 1)
  sph_i = spi * (2.0 * vs_{n+2*i+1} - 1.0) // uniformly distributed in (-π, π)
  vs_{3*i} = cos(sph_i) * sqrt(1.0 - scostheta2)
  vs_{3*i+1} = sin(sph_i) * sqrt(1.0 - scostheta2)
  vs_{3*i+2} = scostheta
end do
return
@#endif
end
```

## 5 Cosine distribution

Generate random points with a cosine distribution on a three-dimensional unit sphere. This distribution is biased  $\cos\theta$  relative to the isotropic distribution, where  $\theta$  is the angle measured from the  $z$  axis. This is the distribution of directions for thermal particles coming off a surface—the factor of  $\cos\theta$  accounting for the solid angle subtended by a surface element.

If particles are emitted into a volume from its enclosing surface uniformly in position and with this distribution in direction, then each point in the volume is visited equally often and each direction is sampled uniformly.

The probability of a particle lying in  $[\theta, \theta + d\theta]$  is  $\frac{1}{2} \sin\theta d\theta$  for  $\theta$  in  $[-\frac{1}{2}\pi, \frac{1}{2}\pi]$  in the isotropic case. For the cosine distribution the probability is  $2 \cos\theta \sin\theta d\theta$  for  $\theta$  in  $[0, \frac{1}{2}\pi]$ . The corresponding distribution in  $z$  is  $f(z) = 2z$  for  $0 \leq z \leq 1$ , and the cumulative distribution is  $F(z) = z^2$ . We can generate this distribution using  $Z = F^{-1}(U) = \sqrt{U}$  where  $U$  is a uniformly distributed random number. The standard “trick” to calculate  $\sqrt{U}$  is to take  $\max(U_1, U_2)$ . However, once again, it seems to be of marginal value to resort to these tricks. Having computed  $z$ , we obtain  $x$  and  $y$  using the same method as for the isotropic distribution.

Because of the way the uniform random numbers are calculated, this routine will never return 0 for any of the components of the position. (However, it *may* return 1 for any of the components.)

See the comments about the dimensioning of  $v$  in the discussion of *random\_isodist*.

<Functions and Subroutines 3> +≡

```

subroutine random_cosdist(v, n, rn_dummy(x))
  implicit_none_f77
  implicit_none_f90
  rn_decl(x) // RNG State
  integer n // Input
  real v0:3*n-1 // Output
  integer i // Local
  real pi, costheta2, phi
  external random_array // External
  data pi/const(3.14159265358979323846264338328)/
  @if  $\neg$ HIPREC

    single_precision vs0:2*n-1 // Output
    single_precision spi, scostheta2, sphi // Local
    external srandom_array
    data spi/3.14159265358979323846264338328/
  @endif

  @if HIPREC

    entry srandom_cosdist(v, n, rn_dummy(x))
  @endif

  if (n ≤ 0)
    return
  call random_array(vn, 2 * n, rn_args(x))
  do i = 0, n - 1
    costheta2 = vn+2*i // cos2 θ
    phi = pi * (two * vn+2*i+1 - one) // uniformly distributed in (−π, π)
    v3*i = cos(phi) * sqrt(one - costheta2)
    v3*i+1 = sin(phi) * sqrt(one - costheta2)

```

```

         $v_{3*i+2} = \text{sqrt}(\text{costheta2})$  // distributed as  $z^2$  in  $(0, 1)$ 
    end do

    return

@if -HIPREC

entry srandom_cosdist(vs, n, rn_dummy(x))
    if (n ≤ 0)
        return
    call srandom_array(vs_n, 2 * n, rn_args(x))
    do i = 0, n - 1
         $\text{scostheta2} = \text{vs}_{n+2*i}$  //  $\cos^2 \theta$ 
         $\text{sphi} = \text{spi} * (2.0 * \text{vs}_{n+2*i+1} - 1.0)$  // uniformly distributed in  $(-\pi, \pi)$ 
         $\text{vs}_{3*i} = \cos(\text{sphi}) * \text{sqrt}(1.0 - \text{scostheta2})$ 
         $\text{vs}_{3*i+1} = \sin(\text{sphi}) * \text{sqrt}(1.0 - \text{scostheta2})$ 
         $\text{vs}_{3*i+2} = \text{sqrt}(\text{scostheta2})$  // distributed as  $z^2$  in  $(0, 1)$ 
    end do

    return
@endif
end

```



**6 INDEX**

*const*: 3, 4, 5.  
*cos*: 3, 4, 5.  
*costheta*: 4.  
*costheta2*: 5.

*FILE*: 1.

*HIPREC*: 3, 4, 5.

*i*: 3, 4, 5.

*implicit\_none\_f77*: 3, 4, 5.

*implicit\_none\_f90*: 3, 4, 5.

*int*: 3.

*log*: 3.

*mod*: 3.

*n*: 3, 4, 5.

*one*: 3, 4, 5.

*phi*: 4, 5.

*pi*: 3, 4, 5.

*ran\_array*: 2.

*ran\_index*: 2.

*random*: 3.

*random\_array*: 3, 4, 5.

*random\_cosdist*: 2, 4, 5.

*random\_gauss*: 2, 3.

*random\_init*: 2.

*random\_isodist*: 2, 4, 5.

*rn\_args*: 2, 3, 4, 5.

*rn\_decl*: 3, 4, 5.

*rn\_dummy*: 3, 4, 5.

*scostheta*: 4.

*scostheta2*: 5.

*sin*: 3, 4, 5.

*single\_precision*: 3, 4, 5.

*sphi*: 4, 5.

*spi*: 3, 4, 5.

*sqrt*: 3, 4, 5.

*srandom*: 3.

*srandom\_array*: 3, 4, 5.

*srandom\_cosdist*: 5.

*srandom\_gauss*: 3.

*srandom\_isodist*: 4.

*stheta*: 3.

*sz*: 3.

*tag*: 2.

*theta*: 3.

*two*: 3, 4, 5.

*v*: 4, 5.

*vs*: 4, 5.

*y*: 3.

*ys*: 3.

*z*: 3.

⟨Functions and Subroutines 3, 4, 5⟩ Used in section 2.

**COMMAND LINE:** "fweave -f -i! -W[ -ykw600 -ytw40000 -j -n/  
/u/karney/degas2/src/randother.web".

**WEB FILE:** "/u/karney/degas2/src/randother.web".

**CHANGE FILE:** (none).

**GLOBAL LANGUAGE:** FORTRAN.