`other random distributions`

September 29, 1999
10:24

Contents

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 $v_{3,\ n}$.

call *random_cosdist*(v, n, *rn_args*(*tag*)) returns a n points cosine-distributed on a sphere in the array $v_{3,\ n}$, 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 ≤ 0)
    return

  call random_array(y, n, rn_args(x))
  do i = 0, int(n / 2) ∗ 2 − 1, 2
    theta = pi ∗ (two ∗ y_i − one)    // uniformly distributed in (−π, π)
    z = sqrt(−two ∗ log(y_{i+1}))    // √(−2 log S)
    y_i = z ∗ cos(theta)
    y_{i+1} = z ∗ sin(theta)
  end do

  if (mod(n, 2) ≡ 0)
    return
  theta = pi ∗ (two ∗ y_{n−1} − one)    // uniformly distributed in (−π, π)
```

```
      z = sqrt(−two ∗ random(rn_args(x)))     // √(−2 log S)
      y_{n−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 ∗ ys_i − 1.0)     // uniformly distributed in (−π, π)
        sz = sqrt(−2.0 ∗ log(ys_{i+1}))     // √(−2 log S)
        ys_i = sz ∗ cos(stheta)
        ys_{i+1} = sz ∗ sin(stheta)
      end do

      if (mod(n, 2) ≡ 0)
        return
      stheta = spi ∗ (2.0 ∗ ys_{n−1} − 1.0)     // uniformly distributed in (−π, π)
      sz = sqrt(−2.0 ∗ srandom(rn_args(x)))     // √(−2 log S)
      ys_{n−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 v_{0:3*n-1}    // Output
      integer i    // Local
      real pi, costheta, phi
      external random_array    // External
      data pi/const(3.14159265358979323846264338328)/
  @#if ¬HIPREC
      single_precision vs_{0:3*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(v_n, 2 * n, rn_args(x))
      do i = 0, n − 1
        costheta = two * v_{n+2*i} − one    // uniformly distributed in (−1, 1)
        phi = pi * (two * v_{n+2*i+1} − one)    // uniformly distributed in (−π, π)
        v_{3*i} = cos(phi) * sqrt(one − costheta^2)
        v_{3*i+1} = sin(phi) * sqrt(one − costheta^2)
        v_{3*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)
   sphi = spi * (2.0 * vs_{n+2*i+1} - 1.0)     // uniformly distributed in (-π, π)
   vs_{3*i} = cos(sphi) * sqrt(1.0 - scostheta^2)
   vs_{3*i+1} = sin(sphi) * sqrt(1.0 - scostheta^2)
   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 θ 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 probablilty of a particle lying in $[\theta, \theta + d\theta]$ is $\frac{1}{2}\sin\theta\, d\theta$ for θ 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 θ in $[0, \frac{1}{2}\pi]$. The corresponding distribution in z is $f(z) = 2z$ for $0 \le z \le 1$, and the cumulative distribution in $F(z) = z^2$. 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 v_{0:3*n-1}   // Output
     integer i   // Local
     real pi, costheta2, phi
     external random_array   // External
     data pi/const(3.14159265358979323846264338328)/
 @#if ¬HIPREC

     single_precision vs_{0:2*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(v_n, 2 * n, rn_args(x))
     do i = 0, n − 1
       costheta2 = v_{n+2*i}   // cos² θ
       phi = pi * (two * v_{n+2*i+1} − one)   // uniformly distributed in (−π, π)
       v_{3*i} = cos(phi) * sqrt(one − costheta2)
       v_{3*i+1} = sin(phi) * sqrt(one − costheta2)
```

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

return

@#if $\neg \mathit{HIPREC}$

entry $\mathit{srandom\_cosdist}(\mathit{vs},\ n,\ \mathit{rn\_dummy}(x))$
if $(n \leq 0)$
return
call $\mathit{srandom\_array}(\mathit{vs}_n,\ 2*n,\ \mathit{rn\_args}(x))$
do $i = 0,\ n-1$
$\mathit{scostheta2} = \mathit{vs}_{n+2*i}$ // $\cos^2\theta$
$\mathit{sphi} = \mathit{spi} * (2.0 * \mathit{vs}_{n+2*i+1} - 1.0)$ // uniformly distributed in $(-\pi,\pi)$
$\mathit{vs}_{3*i} = \cos(\mathit{sphi}) * \mathrm{sqrt}(1.0 - \mathit{scostheta2})$
$\mathit{vs}_{3*i+1} = \sin(\mathit{sphi}) * \mathrm{sqrt}(1.0 - \mathit{scostheta2})$
$\mathit{vs}_{3*i+2} = \mathrm{sqrt}(\mathit{scostheta2})$ // distributed as z^2 in $(0,1)$
end do

return
@#endif
end

6 INDEX

⟨ 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.