MAX-PLANCK-INSTITUT FÜR PLASMAPHYSIK

GARCHING BEI MÜNCHEN

GAUSSF - A Vector Function Version of a Gaussian Random Number Generator for the CRAY-1(R)

Alan Michael McKenney

IPP 6/214 August 1982

Abstract:

GAUSSF is a FORTRAN-callable vector function for the CRAY-1(R) which generates random numbers with a normal (Gaussian) distribution. Its presence in an innermost Do-loop does not inhibit vectorization; moreover, when called from within a vectorized loop, it is over 7 times as fast as the equivalent FORTRAN version, otherwise it is over 4 times as fast.

Die nachstehende Arbeit wurde im Rahmen des Vertrages zwischen dem Max-Planck-Institut für Plasmaphysik und der Europäischen Atomgemeinschaft über die Zusammenarbeit auf dem Gebiete der Plasmaphysik durchgeführt.

Introduction:

In the course of modifying a Monte-Carlo program to run faster on the CRAY-1, it was necessary to rewrite the Gaussian random number generator as a CRAY-1 vector function in CAL (CRAY Assembler Language). I believe that others will find it useful as well.

Method:

Two random numbers ζ_1, ζ_2 with a Gaussian distribution are generated from two random numbers ξ_1, ξ_2 with a uniform distribution between 0 and 1 according to the formulae:*

$$\zeta_1 = \cos(2\pi\xi_1)\ \sqrt{-\ 2\ \log\xi_2}$$

$$\zeta_2 = \sin(2\pi\xi_1)\ \sqrt{-\ 2\ \log\xi_2}$$

Implementation:

Typical FORTRAN Implementation (not vectorized):

When this method of generating Gaussian random numbers is implemented in FORTRAN, the first call causes the computer's random number generator (RANF on the CRAY) to be called to generate two uniformly distributed random numbers, which are used to compute two Gaussian random numbers as previously described, the one is returned and the other saved. The second call causes the saved Gaussian

* Ermakow, S.M. Die Monte-Carlo Methode und verwandte Fragen, R. Oldenbourg Verlag, München, 1975, p. 58

random number to be returned and a flag set that the number has been used. The third call causes a new pair to be computed and one number to be saved as by the first call, and so on. This does not take advantage of the CRAY-1 vector hardware.

GAUSSF Implementation:

GAUSSF maintains a 256-word table of Gaussian random numbers from which random numbers are taken as necessary. The first time GAUSSF is called and every time less than 128 unused numbers remain, the vector version of RANF is called to generate 64 pairs of uniformly distributed random numbers, and the vector hardware and vector versions of SQRT, ALOG, SIN and COS are used to compute 64 pairs of Gaussians using the previously described method, which are then stored in the table; this takes maximum advantage of the vector hardware. Each scalar call to GAUSSF causes the next unused Gaussian random number to be fetched from the table, and, if necessary, another 64 pairs of numbers to be computed. Each vector call causes the next 1 - 64 random numbers to be fetched, and, if necessary, more numbers computed.*

Use:

GAUSSF is written in CAL (CRAY Assembly Language), hence it cannot be compiled with the FORTRAN program which calls it, but must be assembled separately.

GAUSSF is called as a normal single-precision FORTRAN function with one or no arguments. Example:

```
VALUE = GAUSSF()* FACTOR
```

or (equivalently):

```
VALUE = GAUSSF(DUMMY)* FACTOR
```

---

* Cf. the method used in RANF, Library Reference Manual (SR-0014), Cray Research, Inc., Mendota Heights, MN, USA, 1980, p. A-15 ff.

If loops containing GAUSSF calls need not be vectorized, this is sufficient, however, calls to GAUSSF will inhibit vectorization of any loops in which they appear. To avoid this, the CFT compiler directive "VFUNCTION"* may be used:

```
CDIR$ VFUNCTION GAUSSF
        .
        .
        .
      DO 100 J = 1,N
      VALUE(J) = GAUSSF()
100   CONTINUE
```

Example of a job deck:

```
      JOB (parameters)
      CAL.
      CFT.
      LDR.
      /EOF
        .
        .  GAUSSF source.
        .

      /EOF
        .
        .  program
        .
      SUBROUTINE CALLGAUS
      CDIR$ VFUNCTION GAUSSF
            DIMENSION  VAL1(1000), VAL2(1000)
        .
        .  program
        .
           DO 100 J = 1,100
             VAL1(J) = GAUSSF()
```

*FORTRAN(CFT) Reference Manual (SR-0011) Cray Research, Inc., 1980, Part 3, p. 1-18.

```
          VAL2(J) = GAUSSF()
100       CONTINUE
             .
             .   more program
             .
          /EOF
             .
             .
             .
           input data
          /EOF
```

In this job, GAUSSF is assembled, the user's program compiled and run with GAUSSF. In the subroutine "CALLGAUS", GAUSSF has been declared as a vector function, hence the "DO 100" loop will be vectorized. This means that, instead of executing all the operations in the loop once for J = 1, then for J = 2, and so on, in this case, calling GAUSSF for VAL 1(1), then once for VAL 2(1), then once for VAL 1(2), etc., each operation will be executed simultaneously for between one and 64 values of J; in this case. GAUSSF will be called once for VAL 1(1) through VAL 1(64), then once for VAL 2(1) through VAL 2(64), then (repeating the loop) VAL 1(65) through VAL 1(128), etc.

Warning: If GAUSSF is declared as a vector function and called with an argument, CFT assumes that its value is a function of its argument only. Thus, under some circumstances, if it is called from within a loop with an invariant or scalar argument, the call is removed from the loop. The user should therefore always call GAUSSF with no arguments or with a vector (not invariant) dummy argument.

Limitations:

Because of the effective reordering of calculations within a vector loop, loops containing multiple calls to GAUSSF or RANF or calls to both will yield different values, depending on whether the loop is vectorized or not. In the previous example, if the loop is not vectorized, VAL 1(1) will contain the first Gaussian random number and VAL 2(1) the second. If it is vectorized, VAL 1(1) will contain the first and VAL 2(1) the 65th.

Timing:

The times required for 1,000,000 calls were:

	1,000,000 calls
GAUSSF vector loop	.879 sec.
GAUSSF Scalar loop	1.523 sec.
FORTRAN equivalent (Box-Muller method)	6.249 sec.

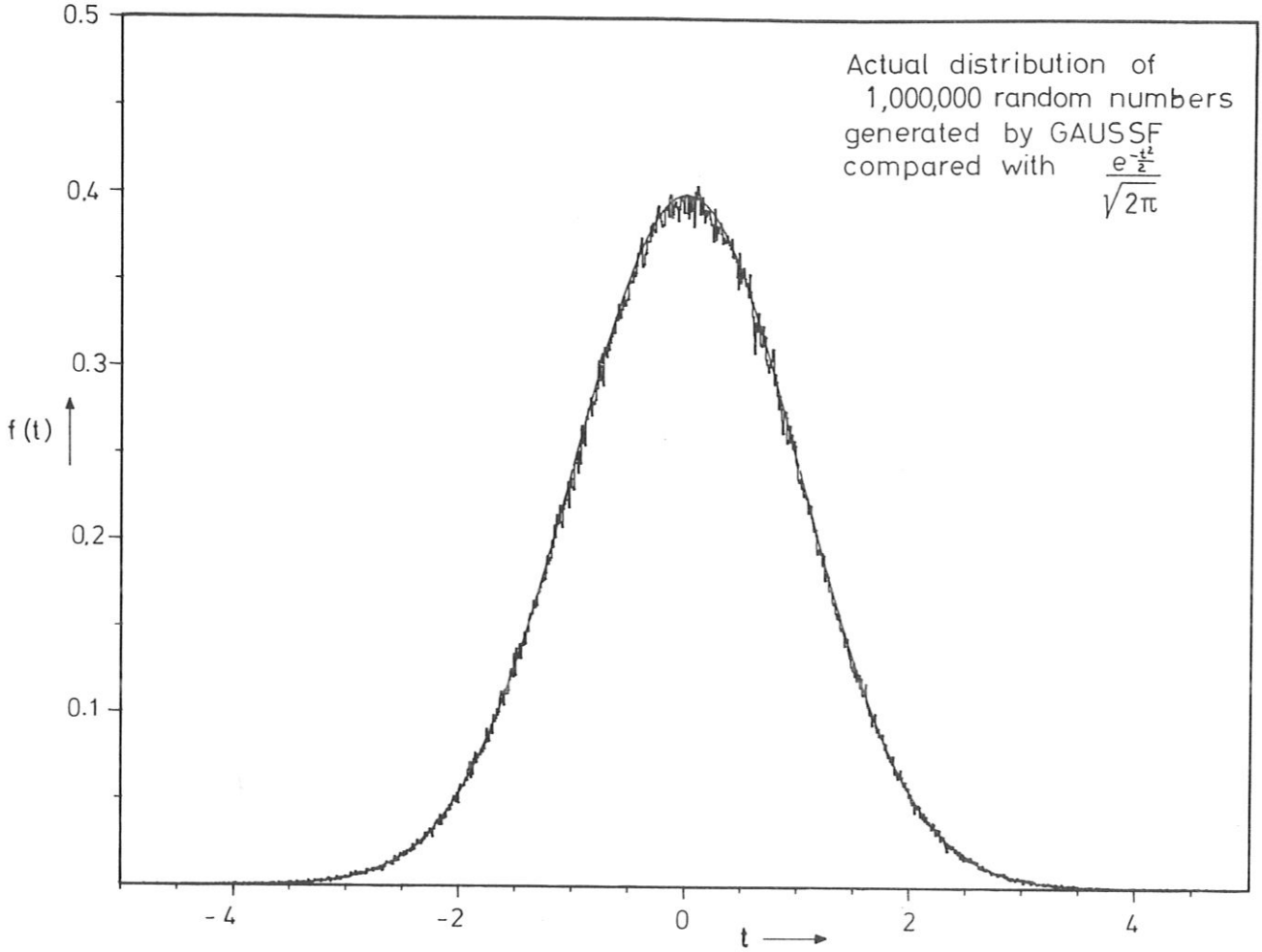
Tests:

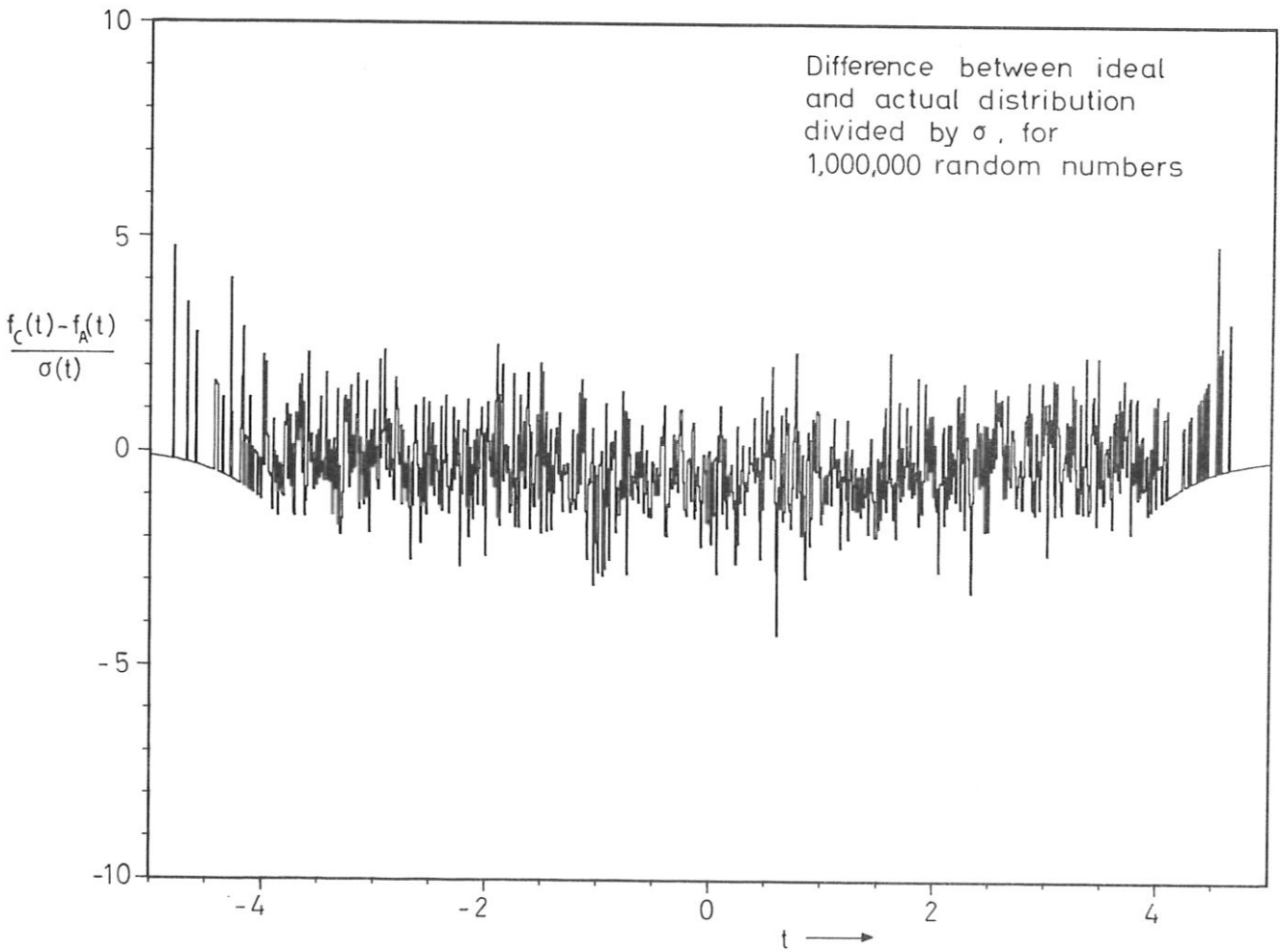
The distribution of the generated numbers $f_c(t)$ was computed by dividing the range $[-5,5]$ into 1,000 "bins" of width $\Delta t = 1/100$, generating N_{total} random numbers, and for each bin j with t-value t_j, computing N_j, the number of random numbers falling within the range $[t_j - \frac{\Delta t}{2}, t_j + \frac{\Delta t}{2})$.

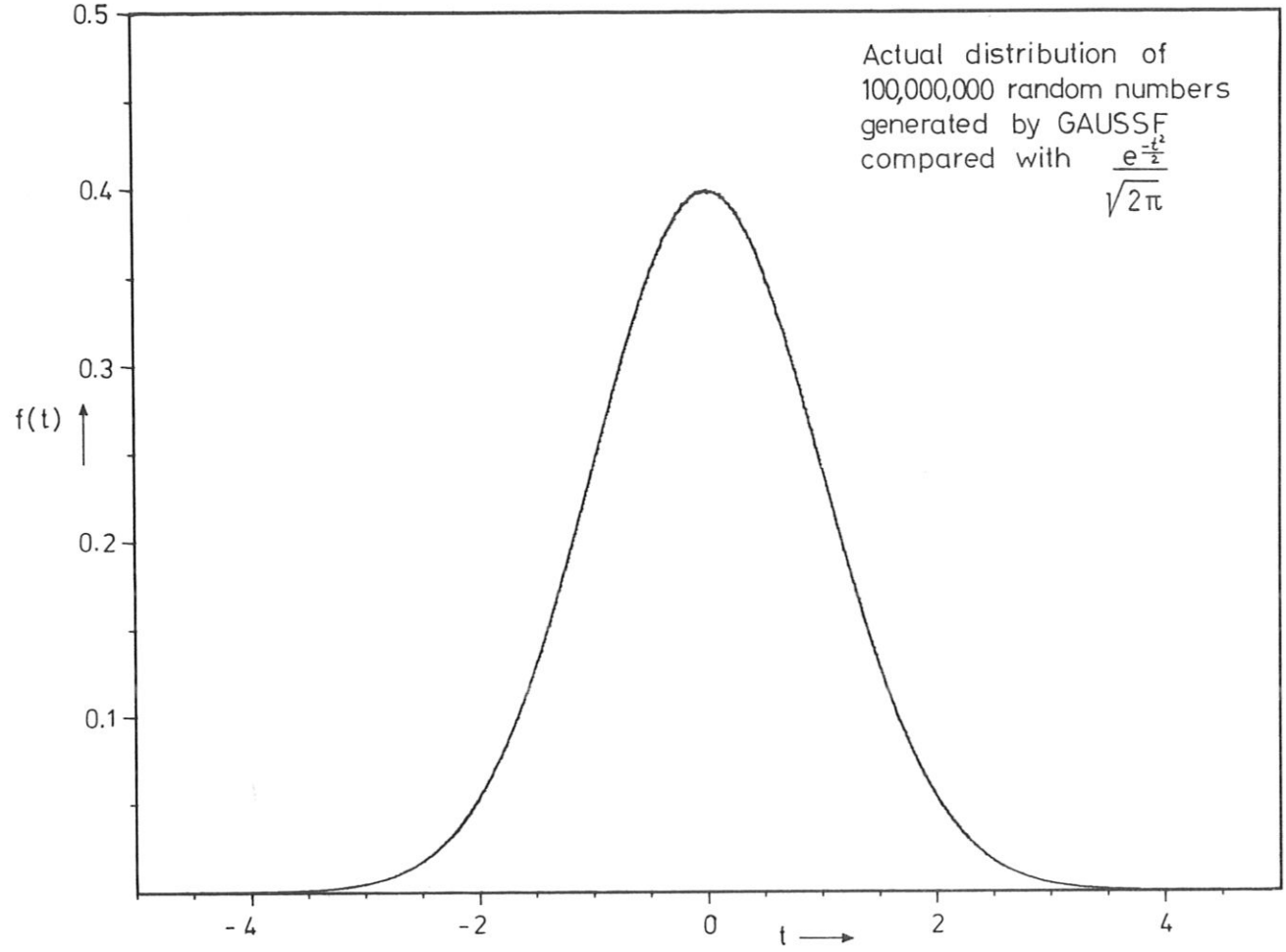
$$f_c(t_j) = \frac{N_j}{\Delta t \cdot N_{total}} \simeq f_A(t_j) = \frac{e^{\frac{-t_j^2}{2}}}{\sqrt{2\pi}}$$

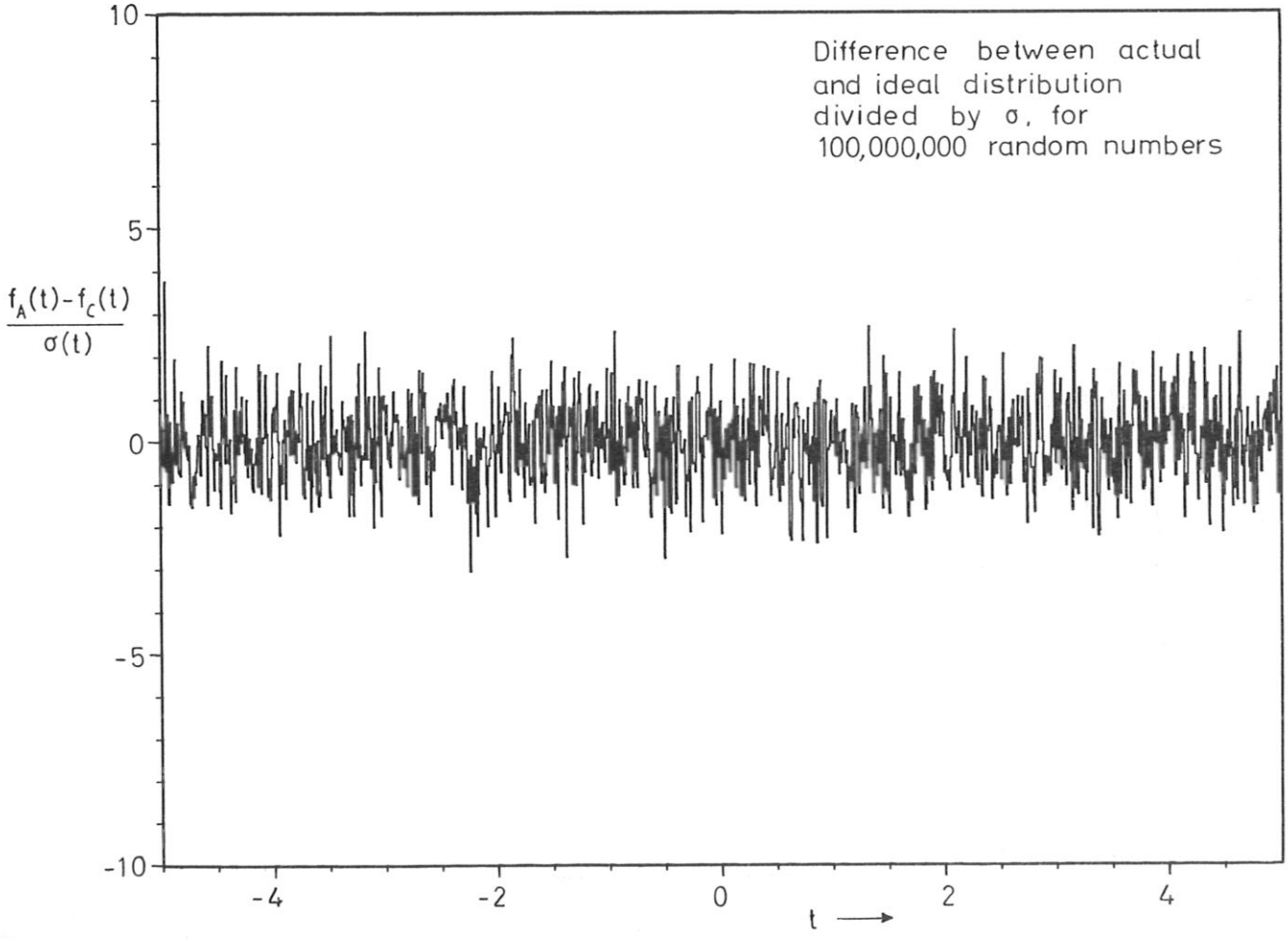
The error, $f_c(t_j) - f_A(t_j)$ was compared with the standard deviation σ:

$$\sigma(t_j) = \sqrt{\frac{f_A(t_j)}{N_{total}} \left(\frac{1}{\Delta t} - f_A(t_j)\right)}$$


Actual distribution of
1,000,000 random numbers
generated by GAUSSF
compared with $\frac{e^{-\frac{t^2}{2}}}{\sqrt{2\pi}}$
0.5
0.4
0.3
0.2
0.1
f (t)
-4
-2
0
2
4
t


Difference between ideal
and actual distribution
divided by σ, for
1,000,000 random numbers
10
5
0
-5
-10
$\frac{f_C(t)-f_A(t)}{\sigma(t)}$
-4
-2
0
2
4
t


Actual distribution of
100,000,000 random numbers
generated by GAUSSF
compared with $\frac{e^{\frac{-t^2}{2}}}{\sqrt{2\pi}}$
0.5
0.4
0.3
0.2
0.1
f(t)
-4
-2
0
2
4
t


Difference between actual
and ideal distribution
divided by σ, for
100,000,000 random numbers
10
5
0
-5
-10
$\frac{f_A(t)-f_C(t)}{\sigma(t)}$
-4
-2
0
2
4
t