Random Flow Generation Procedure Technical Manual

Andrei Smirnov

August 27, 2004

Contents

1 Method

The Random Flow Generation (RFG) procedure provides a random divergence-free vector field, which can also satisfy extra conditions of anisotropy and inhomogeneity. The procedure can be used to generate inflow and initial conditions for LES (large-eddy simulations), and to serve as a subgrid-turbulence model in applications of particle-laden flows. The method has also potential in applications involving acoustic and electromagnetic fields. A detailed description of the method can be found in [7], and some of it's applications are described in [6, 4, 5, 8, 1].

2 Implementation

Figure 1 shows the main flow-chart of the algorithm. It consists of two functions: (1) spectral generator (*genspec*) and vector-field generator (*genvec*). User supplies the input data in form of the size of a spectral sample (integer parameter, *nspec*), correlation tensor for the vector field and time scale. In case of a turbulent flow modeling this should be the velocity correlation tensor, $\overline{u_i u_j}$ and turbulent time scale, τ. In the case of homogeneous turbulence the correlation tensor and time-scale are constant and are suppled as arguments to the *genspec* function (link A in the figure). In the case of an inhomogeneous turbulence these parameters are supplied to the *genvec* function (link B in the figure). Both cases are implemented in the two different versions of the algorithm (see Sec.3.1). Most of the routines are written in a C language with some auxiliary Fortran routines imported from the *Netlib* library (www.netlib.org).

User input parameters to the algorithm are shown in Fig.1 and include: *nspec* - number of spectral modes, 6-dimensional array of velocity correlations, UU (for anisotropic inhomogeneous case: $rfg0$), turbulence time scale, $TURB\_TIME$, turbulent length scale $TURB\_LENGTH$ (for isotropic homogeneous case), and space-time coordinates, x,t. At the beginning there should be only one call to *genspec* function, which initializes the spectral data. These consists of four arrays: *Omega*, $U1$, $U2$, and K. While the first one is the array of scalar values, the other three are all arrays of 3D vectors. These four arrays are global variables and after they are initialized by *genspec* they are repeatedly used by the *genvec* function to generate the random vector field. In the homogeneous isotropic version of the algorithm ($rfg0$) the user supplies the *nspec*, $TURPB\_TIME$, and $TURB\_LENGHT$ parameters to *genspec* function, and then repeatedly calls the *genvel* functions for different spatial locations and time. In the inhomogeneous

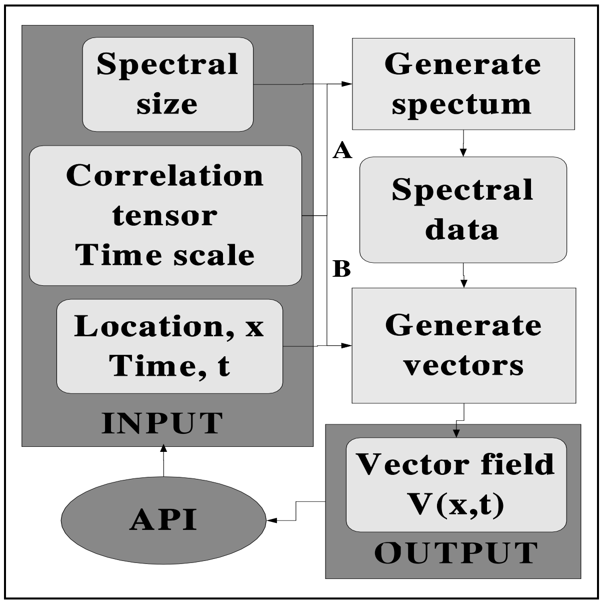

Figure 1: Application programming interface for RFG algorithm

anisotropic case ($rfg1$), the user calls *genspec* with only one parameter, defining the size of the spectral sample (*nspec*), and then repeatedly calls *genvec* with for different x, t, as well as different values of velocity correlation tensor and time scale. In this way one provides spacial inhomogeneity of the generated vector field. The anisotropy is determined by the correlation tensor.

The complete listing of the main RFG routine is given in the Appendix.

3 User's Guide

3.1 Distribution

The main distribution of RFG is located at http://cfd.mae.wvu.edu/rfg. The program was developed and tested on a Linux PC, under RedHat OS. There are two versions of the algorithm: (1) homogeneous isotropic case, and (2) inhomogeneous anisotropic case. The first version simply reproduces the method of Kraichnan [2], and does not include any *Netlib* routines. The second version can generate anisotropic inhomogeneous vector fields with prescribed anisotropy, and inhomogeneity. It makes use of several *Netlib* routines, which retrieve diagonal components of the velocity correlation tensor, and handle coordinate transformations in and out of coordinate system aligned with the tensor's principal axes. The second version thus consists of both RFG routines (C) and *Netlib* routines (Fortran). The distributions contain the following files:

- Homogeneous isotropic case:

```
makefile      # Makefile
rfg.h         # RFG header file
vecalg.h      # Vector algebra macros
gauss.c       # Gaussian random number generator
rfg.c         # RFG routine
random.f      # Random number generator
rfgtest.f     # RFG test
```

- Inhomogeneous anisotropic case:

```
makefile       # Makefile
rfg.h          # RFG header file
vecalg.h       # Vector algebra macros
diag.c         # Diagonalization routines
gauss.c        # Gaussian random number generator
rfg.c          # RFG routine
phytag.f       # Auxiliary Netlib routine
random.f       # Random number generator
rfgtest.f      # RFG test
tql1.f         # Auxiliary Netlib routine
tql2.f         # Auxiliary Netlib routine
tred1.f        # Auxiliary Netlib routine
tred2.f        # Auxiliary Netlib routine
```

In addition to this both distributions include README files containing the compilation instructions.

3.2 Installation

The source files should be unpacked into an empty directory with a command:

```
tar xvzf rfg1.tgz
```

or on any Unix machine a more general command would be:

```
gunzip rfg1.tgz -c | tar xvf -
```

Since RFG routines are supposed to be integrated into other programs, and example main program was written in Fortran, containing a simple test of the RFG routine. The program file is *rfgtest.f*. To compile this file together with other files source files into an executable one should issue the *make* command, like this:

```
make
```

This will create the executable *rfgtest*. The purpose of the test is to run a *genspec* function at the beginning and run the *genvec* function repeatedly to generate random vector field. To test the executable issue the command:

```
./rfgtest
```

This will produce a long output of numbers. The first 10 lines may look like this:

```
  0.   0. -1.04623053  0.315150145  0.318838617
  0.   1.  0.675930783 -0.0619834219 -0.0272438827
  0.   2.  1.27071291 -1.18061539 -0.18484001
  0.   3.  0.947634502 -2.32104215  0.074804578
  0.   4.  0.775465738 -2.30559589  0.522083564
  0.   5.  0.575862596 -0.912436065  0.766931245
  0.   6. -0.175024857  0.511847868  0.686787079
  0.   7. -0.974941132  0.781299244  0.500546874
  0.   8. -1.42777037  0.492852612  0.556896856
  0.   9. -1.98023326  0.698884086  1.05550196
```

The *rfgtest.f* file contains further instructions on how to view the results using the *gnuplot* plotting program.

3.3 Usage

The example given in the rfgtest.f file illustrates setting up and usage of the RFG routine. Even though the RFG routines are written in C-language, the example was prepared in Fortran to illustrate how to use RFG together with Fortran-based codes.

It can be seen from the example that the external data supplied by the user include the spectral sample size, *nspec*, the velocity correlation tensor, as well as the turbulence time-scale, which can be given as functions of space and time. The *genspec* routine serves as an initializer and should only be called once. After that the *genvec* routine can be called many times for different x, t coordinates and it will return the values of a random vector for each x, t pair.

Increasing the value of the parameter nspec (the argument of the genspec function) will increase the spectral resolution of the algorithm and decrease the speed of execution. Thus, this value should be selected as a trade-off between the two. For debugging purposes a value of 10 can be reasonable, while for production runs the value of 100 or higher may be appropriate.

4 Acknowledgments

This work has been performed under a DOD EPSCoR project sponsored by the Office of Naval Research (ONR), Grant No.N00014-98-1-0611, monitored by Dr. Edwin P. Rood and Patrick L. Purtell.

References

[1] I. Celik, A. Smirnov, and S. Shi. Les of bubble dynamics in wake flows. In *Twenty-Fourth Symposium on Naval Hydrodynamics*, volume 2, pages 219–233, Val de Reuil, Japan, 2002.

[2] R.H. Kraichnan. Diffusion by a random velocity field. *Phys. Fluid*, 11:43–62, 1970.

[3] A. Li, G. Ahmadi, R.G. Bayer, and M.A. Gaynes. Aerosol particle deposition in an obstructed turbulent duct flow. *J. Aerosol Sci.*, 25(1):91–112, 1994.

[4] S. Shi, A. Smirnov, and I. Celik. Large eddy simulations of particle-laden turbulent wakes using a random flow generation technique. In *ONR 2000 Free*

Surface Turbulence and Bubbly Flows Workshop, pages 13.1–13.7, California Institute of Technology, Pasadena, CA, 2000.

[5] S. Shi, A. Smirnov, and I. Celik. A new approach for generating time-dependent inflow boundary with application to large-eddy simulation of flat plate wake. In *International Mechancial Engineering Congress and Exposition*, Orlando, FL, 2000.

[6] A. Smirnov, S. Shi, and I. Celik. Random Flow Simulations with a Bubble Dynamics Model. In *ASME Fluids Engineering Division Summer Meeting*, number 11215 in FEDSM2000, Boston, MA, 2000.

[7] A. Smirnov, S. Shi, and I. Celik. Random flow generation technique for large eddy simulations and particle-dynamics modeling. *Trans. ASME. Journal of Fluids Engineering*, 123:359–371, 2001.

[8] A.V. Smirnov. Flame propagation simulations with random flow generation model. In *Micro-Mixing in Turbulent Reactive Flows*, pages 100–104. Torus Press, Moscow, Russia, 2004.

A Appendix: RFG Main Routines

A.1 Homogeneous Isotropic Case

The equations numbers refer to the work of Ahmadi et.al [3].

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "vecalg.h"
#include "rfg.h"

#ifndef SMALL
#define SMALL   1.e-30
#endif

int   ne = 1; /* Number of terms in the series Eq.(15) */

REAL
   *Omega, /* Eq.15 */
   *U1,*U2, /* velocity vectors (Eqs.15-17) */
   *K; /* wave vectors (Eqs.15-17) */

void    Allocate(REAL **A, int n)
{
   if ((*A = (REAL *) malloc(sizeof(**A)*n)) == NULL)
   {
      fprintf(stderr,"CAN'T ALLOCATE MEMORY\n");
      exit(1);
   }
}
void   genspec_
(
   int   *Ne,
   REAL   *TURB_TIME,
   REAL   *TURB_LENGTH
)
/*
```

```
   Generate spectral expansion coefficients
*/
{
   extern   REAL   gauss_(); /* get a Gaussian random variable */
   extern   void   gaussn_(REAL *, REAL, int); /* get an array of
                                           Gaussian random numbers */
   int   ie;
   REAL   fe,turb_time=*TURB_TIME;
   ne=*Ne;
   if (ne<=0) return;
   fe=sqrt(2./(REAL)ne);
   Allocate(&Omega,ne);
   Allocate(&K ,ne*DIM);
   Allocate(&U1,ne*DIM);
   Allocate(&U2,ne*DIM);
   seed_(); /* initialize the random generator */
   gaussn_(K,.5,ne*DIM);
   for (ie=0; ie<ne; ie++)
   {   int   i,j=ie*DIM;
      REAL
         a,V1[DIM],V2[DIM], /* random vectors xi and zeta
                              (Eq.16) */
         *u1=U1+j,*u2=U2+j,*k=K+j;
      Omega[ie] = gauss_()/turb_time;
      gaussn_(V1, fe, DIM);
      gaussn_(V2, fe, DIM);
      VECP(u1,V1,k); /* Eq.16 */
      VECP(u2,V2,k);
      for (i=0; i<DIM; i++)k[i]/=TURB_LENGTH[i];
   }
}
void   delspec_()
{
   free(Omega);
   free(K);
   free(U1);
   free(U2);
}
```

```
void   genvec_
(
//   INPUT:
   REAL   *t,  // time
   REAL   *x,  // coordinates
// OUTPUT:
   REAL   *v   // velocities
)
{
   int   i,ie;
   REAL   a,c,s;
   for (i=0; i<DIM; i++) v[i]=0.0;
   if (ne<=0)  return;
   for (ie=0; ie<ne; ie++)
   {   int n=ie*DIM;
      REAL   *k=K+n;
      a=SCLP(k,x)+Omega[ie]**t;
      c=cos(a); s=sin(a);
      for (i=0; i<DIM; i++)
         v[i]+=U1[n+i]*c+U2[n+i]*s;
   }
}
```

A.2 Inhomogeneous Anisotropic Case

The equations numbers refer to the work of Ahmadi et.al [3].

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "rfg.h"
#include "vecalg.h"

#ifndef SMALL
#define SMALL   1.e-30
#endif

int   ne = 1; /* Number of terms in the series Eq.(15) */
```

```
REAL
   fe = 1.41421,
   *Omega, /* Eq.15 */
   *U1,*U2, /* velocity vectors (Eqs.15-17) */
   *K; /* wave vectors (Eqs.15-17) */

void    Allocate(REAL **A, int n)
{
   if ((*A = (REAL *) malloc(sizeof(**A)*n)) == NULL)
   {
      fprintf(stderr,"CAN'T ALLOCATE MEMORY\n");
      exit(1);
   }
}
void   genspec_
(
   int   *Ne
)
/*
 *   Generate spectral expansion coefficients
 */
{   extern   void   diag
   (
      REAL *A, /* velocity correlations */
      REAL *D  /* diagonal vector after diagonalization of A */
   );
   extern   REAL   gauss_(); /* get a Gaussian random variable */
   extern   void   gaussn_(REAL *, REAL, int); /* get an array of
                                          Gaussian random numbers */
   int   i,ie;
   ne=*Ne;
   if (ne<=0) return;
   fe=sqrt(2./(REAL)ne);
   Allocate(&Omega,ne);
   Allocate(&K ,ne*DIM);
   Allocate(&U1,ne*DIM);
   Allocate(&U2,ne*DIM);
   seed_(); /* initialize the random generator */
//   seed0(999); //same numbers every time
   gaussn_(K,.5,ne*DIM);
   for (ie=0; ie<ne; ie++)
   {   int   j=ie*DIM;
      REAL   a,V1[DIM],V2[DIM], /* random vectors xi and zeta
                                  (Eq.16) */
         *u1=U1+j,
         *u2=U2+j,
         *k=K+j;
      Omega[ie] = gauss_();
      gaussn_(V1, 1., DIM);
      gaussn_(V2, 1., DIM);
      VECP(u1,V1,k); /* Eq.16 */
      VECP(u2,V2,k);
   }
}
void   delspec_()
{
   free(Omega);
   free(K);
   free(U1);
   free(U2);
}
void   genvec_
(
//   INPUT:
   REAL   *t,  // time
   REAL   *x,  // coordinates
   REAL   *TT, // Turbulent Time: scalar
   REAL   *UU, // velocity correlations: UU,UV,VV,UW,VW,WW
// OUTPUT:
   REAL   *v   // velocities
)
{   extern   void   diag
   (
      REAL *A, /* velocity correlations */
      REAL *D  /* diagonal vector after diagonalization of A */
   );
   int   i,ie;
   REAL   a,c,s,
      d[DIM],dd=0.0,
      turb_time=*TT;

   if (ne<=0)  return;
   diag(UU,d);
   for (i=0; i<DIM; i++)
   {   REAL   r=fabs(d[i]);
      d[i]=sqrt(r);
      dd+=r;
      v[i]=0.0;
   }
   dd=sqrt(dd);
   for (ie=0; ie<ne; ie++)
   {   int n=ie*DIM;
      REAL   k[DIM],
            *u1=U1+n,*u2=U2+n;
      for (i=0; i<DIM; i++)
         k[i]=K[n+i]/(d[i]>SMALL?turb_time*d[i]:turb_time*dd);
      a=SCLP(k,x)+Omega[ie]**t/turb_time;
      c=cos(a); s=sin(a);
      for (i=0; i<DIM; i++)
         v[i]+=u0[i]*c+u2[i]*s;
   }
   for (i=0; i<DIM; i++)v[i]*=fe*d[i];//V-anisotropy
   btrans(v);
}
```