

# TuGen

*Synthetic Turbulence Generator, Manual and User's Guide*

Gilling, Lasse

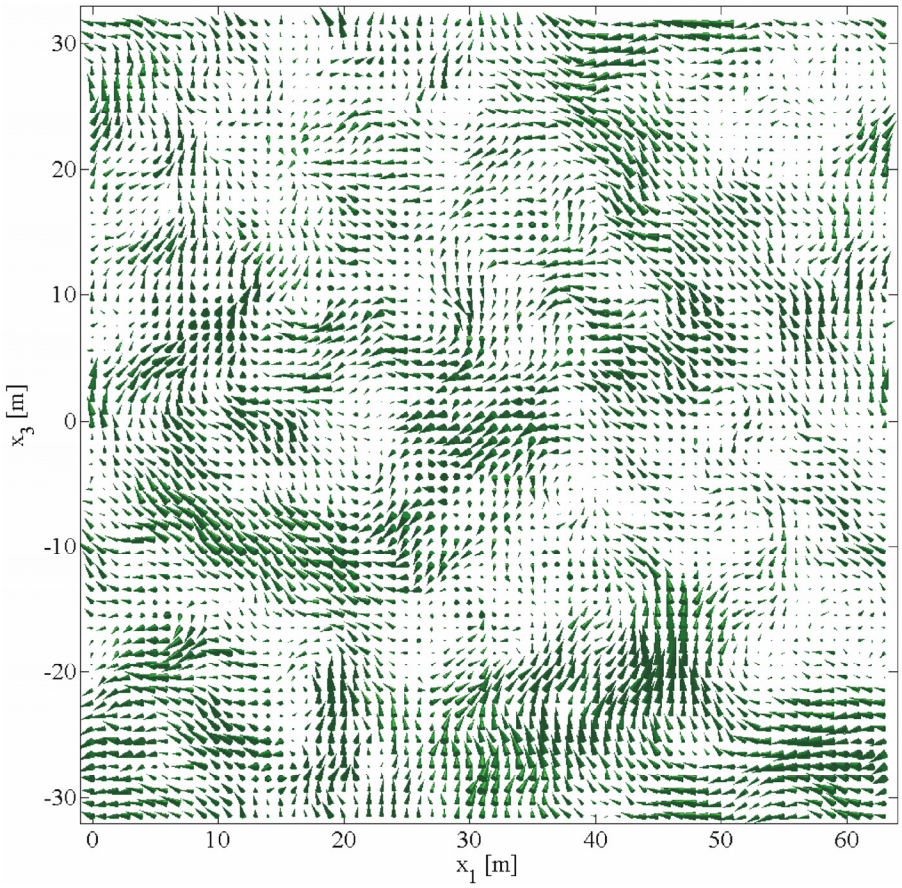
*Publication date:*
2009

*Document Version*
Publisher's PDF, also known as Version of record

# TuGen

## Synthetic Turbulence Generator
## Manual and User's Guide

**Lasse Gilling**




**ISSN 1901-726X**
**DCE Technical Report No. 76**

AALBORG UNIVERSITY
**Department of Civil Engineering**

**ISSN 1901-726X**
**DCE Technical Report No. 76**

Aalborg University
Department of Civil Engineering
Division of Structural Mechanics

**DCE Technical Report No. 76**

# TuGen
## Synthetic Turbulence Generator, Manual and User's Guide

by

Lasse Gilling

September 2009

## Scientific Publications at the Department of Civil Engineering

***Technical Reports*** are published for timely dissemination of research results and scientific work carried out at the Department of Civil Engineering (DCE) at Aalborg University. This medium allows publication of more detailed explanations and results than typically allowed in scientific journals.

***Technical Memoranda*** are produced to enable the preliminary dissemination of scientific work by the personnel of the DCE where such release is deemed to be appropriate. Documents of this kind may be incomplete or temporary versions of papers—or part of continuing work. This should be kept in mind when references are given to publications of this kind.

***Contract Reports*** are produced to report scientific work carried out under contract. Publications of this kind contain confidential matter and are reserved for the sponsors and the DCE. Therefore, Contract Reports are generally not available for public circulation.

***Lecture Notes*** contain material produced by the lecturers at the DCE for educational purposes. This may be scientific notes, lecture books, example problems or manuals for laboratory work, or computer programs developed at the DCE.

***Theses*** are monograms or collections of papers published to report the scientific work carried out at the DCE to obtain a degree as either PhD or Doctor of Technology. The thesis is publicly available after the defence of the degree.

***Latest News*** is published to enable rapid communication of information about scientific work carried out at the DCE. This includes the status of research projects, developments in the laboratories, information about collaborative work and recent research results.

Published 2009 by
Aalborg University
Department of Civil Engineering
Sohngaardsholmsvej 57,
DK-9000 Aalborg, Denmark

Printed in Aalborg at Aalborg University

ISSN 1901-726X
DCE Technical Report No. 76

An implementation of Mann's method of generating synthetic turbulence is described. A brief introduction to the method is given prior to some details on the implementation. Several tests to verify the implementation follows and finally the use of the program is described.

The code is written in Fortran 90 in a very 77-like form. It is printed in Section 5 and available for download from:

http://vbn.aau.dk/fbspretrieve/18432733/Fortran_code_for_download.zip

Please note the comment given in Section 5 on page 15. If you have any comments or questions, please feel free to contact me on my e-mail:

lassegilling@hotmail.com



# Contents

# 1 How it Works

The program is based on the paper by Mann (1998). There are however some points not addressed in the paper, which are necessary to produce real and divergence free turbulence fields. These issues will be discussed in the following.

## 1.1 The Method

The method is based on Fourier transform of wave number vectors. Thereby, the velocity field consist of the sum of a large number of linear waves as the one illustrated in Figure 1. If the physical domain is resolved into $N_1 \times N_2 \times N_3$ points the wind field will consist of the sum of $N_1 \times N_2 \times N_3$ many linear waves.

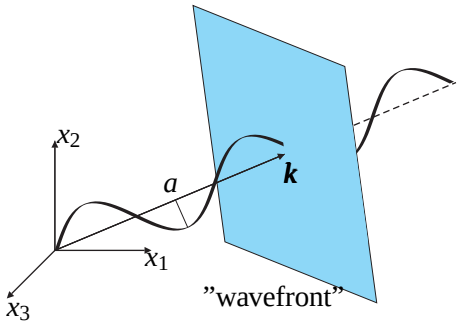



**Figure 1:** A single linear component of the wind field. $\boldsymbol{k}$ is the wave number vector which has the coordinates of the point in wave number space. The direction and wavelength is given by $\boldsymbol{k}$ and the amplitude $a$ is $E(\boldsymbol{k})$ multiplied by a random number.

To introduce the concept a pseudo algorithm is presented in Algorithm 1. In Section 5 the code is printed. Below, the matrices are defined.

**Algorithm 1**: Pseudo algorithm

**for** *all wave number vectors* $\boldsymbol{k}$ *in the domain* **do**
  Determine random vector $\boldsymbol{n}(\boldsymbol{k})$. $\boldsymbol{n}$ must be Gaussian and complex
  Determine matrix $\boldsymbol{B}(\boldsymbol{k})$, c.f. (13) in Mann (1998)
  Determine covariance tensor $\boldsymbol{C}(\boldsymbol{k})$
  Determine Fourier-Stieltje vectors $d\boldsymbol{Z}(\boldsymbol{k}) = \boldsymbol{BC}\boldsymbol{n}$
**end**
Make generated field of $d\boldsymbol{Z}(\boldsymbol{k})$ "symmetric", i.e. $d\boldsymbol{Z}(-\boldsymbol{k}) = d\boldsymbol{Z}^*(\boldsymbol{k})$
Run three dimensional FFT to get velocity field $\boldsymbol{v}(\boldsymbol{x}) = \int e^{i\boldsymbol{k}\cdot\boldsymbol{x}} d\boldsymbol{Z}(\boldsymbol{k})$
Correct divergence if desired
Write velocities to file and/or do diagnostics

The Fourier-Stieltje vectors are determined from

$$d\boldsymbol{Z}(\boldsymbol{k}) = \boldsymbol{BC}\boldsymbol{n} \tag{1}$$

where

$$\boldsymbol{B} = \begin{bmatrix} 1 & 0 & \zeta_1 \\ 0 & 1 & \zeta_2 \\ 0 & 0 & \zeta_3 \end{bmatrix} \tag{2}$$

$$\boldsymbol{C} = \frac{1}{k^2}\sqrt{\frac{E(k)}{4\pi}}\begin{bmatrix} 0 & k_3 & -k_2 \\ -k_3 & 0 & k_1 \\ k_2 & -k_1 & 0 \end{bmatrix} \tag{3}$$

$$\boldsymbol{n} = \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix} \tag{4}$$

$$\zeta_1 = C_1 - \frac{k_2}{k_1}C_2 \qquad , \qquad \zeta_2 = \frac{k_2}{k_1}C_1 + C_2 \qquad , \qquad \zeta_3 = \frac{k_0^2}{k^2} \tag{5}$$

$$C_1 = \frac{\beta k_1^2(k_0^2 - 2k_{3,0}^2 + \beta k_1 k_{3,0})}{k^2(k_1^2 + k_2^2)} \tag{6}$$

$$C_2 = \frac{k_2 k_0^2}{(k_1^2 + k_2^2)^{3/2}} \arctan\left(\frac{\beta k_1\sqrt{k_1^2 + k_2^2}}{k_0^2 - k_{3,0}k_1\beta}\right) \tag{7}$$

$$k_1 = k_{1,0} \qquad , \qquad k_2 = k_{2,0} \qquad , \qquad k_3 = k_{3,0} + \beta k_1 \tag{8}$$

$$k = |\boldsymbol{k}| = \sqrt{k_1^2 + k_2^2 + k_3^2} \qquad , \qquad k_0 = |\boldsymbol{k}_0| = \sqrt{k_1^2 + k_2^2 + k_{3,0}^2} \tag{9}$$

$$\beta = (kL)^{-2/3}\Gamma \tag{10}$$

For $\boldsymbol{k} = \boldsymbol{0}$ (5)-(7) will be undefined. In this point $\boldsymbol{C}$ will be zero so $\boldsymbol{B}$ is chosen as the identity matrix to prevent the program to fail due to division with zero. For $k_1 = 0$ the limit values of $\zeta_1$, $\zeta_2$ and $\zeta_3$ will be used

$$\lim_{k_1 \to 0} \zeta_1 = -\beta \qquad , \qquad \lim_{k_1 \to 0} \zeta_2 = 0 \qquad , \qquad \lim_{k_1 \to 0} \zeta_3 = 1 \tag{11}$$

Each point in the resolved spectral domain is described by a wave number vector $\boldsymbol{k}$. $E(\boldsymbol{k})$ is the energy spectrum, which gives the energy associated with the linear wave defined by the wave number vector $\boldsymbol{k}$

$$E(\boldsymbol{k}) = \alpha\varepsilon^{2/3}L^{5/3}\frac{L^4k^4}{(1 + L^2k^2)^{17/6}} \tag{12}$$

$L$ defines the length scale and $\alpha\varepsilon^{2/3}$ scales the intensity. $\Gamma$ gives the degree of anisotropy. These three parameters should be given as input to the program along with the dimensions of the domain.

To ensure the generated wind field is real the Fourier-Stieltje components must satisfy the condition

$$d\boldsymbol{Z}(-\boldsymbol{k}) = d\boldsymbol{Z}^*(\boldsymbol{k}) \tag{13}$$

Consequently, only about half of the random numbers can be chosen arbitrarily. For some special $\boldsymbol{k}$, e.g. $\boldsymbol{k} = \boldsymbol{0}$, (13) enforces the imaginary part to be zero.

The above equations are based on the assumption that the side lengths of the domain are large compared to the turbulence length scale. Mann (1998) states that the approximation is good for $dx_{(j)}N_{(j)} < 8L$ for $j = 2, 3$, where the parentheses in the index cancels the summation convention.

## 1.2 Periodicity

As the FFT-algorithm assumes the function to be periodic, the generated field will be periodic in all three directions. That is, there will be a correlation coefficient of (nearly) unity of the velocities on each side of the box.

In most applications this is not desired. To remedy this problem, generate a wind field larger than the desired domain and discard some of the data. The required size of the generated wind field depends on the resolution of the domain and the length scale of the turbulence. For wind turbines the largest of $2D$ or $8L$ is recommended, where $D$ is the rotor diameter.

## 1.3 Incompressibility

The components of the wind field are determined from Fourier transformation of the Fourier-Stieltje vectors

$$\boldsymbol{v}(\boldsymbol{x}) = \int e^{i\boldsymbol{k}\cdot\boldsymbol{x}} d\boldsymbol{Z}(\boldsymbol{k}) \tag{14}$$

Incompressibility of the wind field requires zero divergence of the field. Using index notation with the summation convention the condition can be written

$$\frac{\partial v_j}{\partial x_j} = 0 \tag{15}$$

By combining (14) and (15) a condition can be imposed on the Fourier-Stieltje vectors

$$\frac{\partial v_j}{\partial x_j} = i \int e^{i\boldsymbol{k}\cdot\boldsymbol{x}} k_j dZ_j(\boldsymbol{k}) = 0 \quad \Rightarrow \quad \boxed{k_j dZ_j(\boldsymbol{k}) = 0} \tag{16}$$

Pre-multiplication of (1) with $\boldsymbol{k}$ yields

$$\boldsymbol{k}\, d\boldsymbol{Z}(\boldsymbol{k}) = \frac{1}{k^2}\sqrt{\frac{E(k)}{4\pi}} \begin{bmatrix} k_1k_2\zeta_1 - k_2k_3 + k_2^2\zeta_2 + k_2k_3\zeta_3 \\ k_1k_3 - k_1^2\zeta_1 - k_1k_2\zeta_2 - k_1k_3\zeta_3 \\ 0 \end{bmatrix}^T \begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix} \tag{17}$$

If $n_1$ and $n_2$ are independent random variables (17) will only be zero in the isotropic case where $\zeta_1 = \zeta_2 = 0$ and $\zeta_3 = 1$. If $n_1$ and $n_2$ are chosen to satisfy $n_2 = n_1 k_2/k_1$ then (17) will also be zero in the anisotropic case. This is a poor choice, however, as this will give $v_3 = 0$ in all points.

If a divergence free field is required then the divergence may be corrected as described in Section 2.2.

## 1.4 Possible Improvements

Below are some possible improvements listed:

- In the present implementation the entire field of Fourier-Stieltje vectors are generated from independent random numbers. Then, symmetry is enforced as defined in (13). This procedure costs a little on the computation time, but it is easier to change the way symmetry is enforced.

- The divergence correction can implemented by the TDMA algorithm instead of the MINRES algorithm for solving the linear equations which might increase computational speed.
- The approximation for large dimensions of the domain compared to the turbulent length scale can be avoided, but then the spectral tensor must be obtained by numerical integration. This integration could be implemented.

# 2 Additional Features

Below, two features are presented. Both can be used, but they are not required for generating the velocity fields. The first determines the one-dimensional correlation function which can be used to verify or describe the field. It produces extra output but does not modify the generated turbulence field. The latter is a divergence correction method. It produces no extra output, but makes changes to the velocity field.

## 2.1 One-Dimensional Correlation Function

In all three directions of the domain the one-dimensional auto-correlation function is determined for all three components. The correlations are averaged over the remaining two directions.

Further, the average correlation between the three velocity components are determined.

## 2.2 Divergence Correction

For isotropic field the divergence of the continuous field will be zero. This is not the case for the discrete representation, however. As was shown in Section 1.3 anisotropic fields are not divergence free, even in the continuous representation. In this section a method to correct the divergence to zero will be shown.

The corrected velocity field can be determined from

$$\tilde{v}_j(\boldsymbol{x}) = v_j(\boldsymbol{x}) - \frac{\partial p(\boldsymbol{x})}{\partial x_j} \tag{18}$$

where the function $P(\boldsymbol{x})$ is defined from

$$\frac{\partial^2 p}{\partial x_j^2} = \frac{\partial v_j}{\partial x_j} \tag{19}$$

By this definition $\tilde{v}_j(\boldsymbol{x})$ will satisfy (15) identically. The definition of $p$ is recognized as a Poisson differential equation. For a given numerical differential scheme the equation can be solved by solving a system of linear equations. Here and in the code the 2nd order central difference scheme will be used.

The divergence is determined in all points and stored in a vector $\boldsymbol{d}$. It is used as right hand side in (19). The problem is formulated as a linear system $\boldsymbol{d} = \boldsymbol{A}\boldsymbol{p}$, where $\boldsymbol{A}$ is a coefficient matrix defined from the numerical differential scheme. The linear system is solved by the MINRES algorithm, which is possible because $\boldsymbol{A}$ is symmetric.

In the MINRES algorithm matrix-vector multiplications are performed repeatedly. For typical dimensions of the domain the number of points is so large that it would require huge amounts of memory to allocate the full coefficient matrix. If for instance $N_1 = 8192$ and $N_2 = N_3 = 32$ the coefficient matrix should be of dimension $8,388,608 \times 8,388,608$. However, the matrix is very sparse and has a simple structure.

A subroutine has been written that will take any vector as input and give the product with the coefficient matrix $\boldsymbol{A}$ as output. By utilizing the simple structure of the matrix this has been implemented in a way that never forms the matrix it self (not even in some kind of sparse format). The approach has very low demands to memory and is quite fast due to the efficiency of the MINRES algorithm.

# 3 Verifications

To check the implementation of the program several tests have been performed. Below some of the results from these tests are presented.

## 3.1 Vector Plot

In Figure 2 a vector plot of a generated velocity field is shown. Each cone in the figure represents a velocity vector. The sizes of the cones are scaled to show the magnitude of the vectors. It is seen that the field is periodic. Further, it illustrates the spatial correlation as structures can be seen in the field.

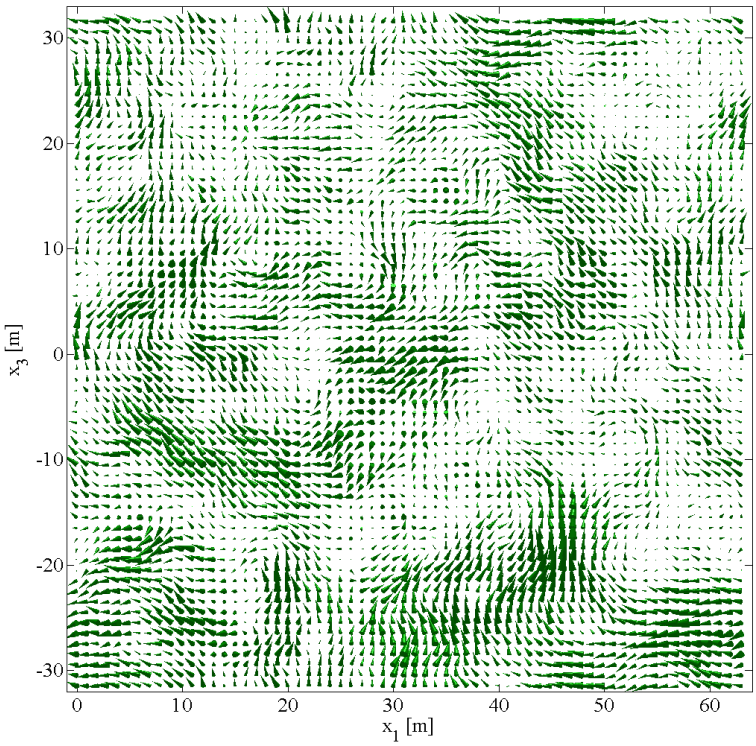



**Figure 2:** Vector plot of a generated velocity field. The field is isotropic with $dx_j = 1\,m$, $N_j = 64$ and $L = 25\,m$.

## 3.2 Isotropy

In Figure 3 the variances of the three velocity components are shown for 250 uncorrelated realizations. The generated field and the domain it self is isotropic with $\Gamma = 0$, $dx_1 = dx_2 = dx_3$ and $N_1 = N_2 = N_3$. Therefore, the average variances of the three components should be identical. It is seen that there are large variations from realization to realization but the average variances are close to identical.

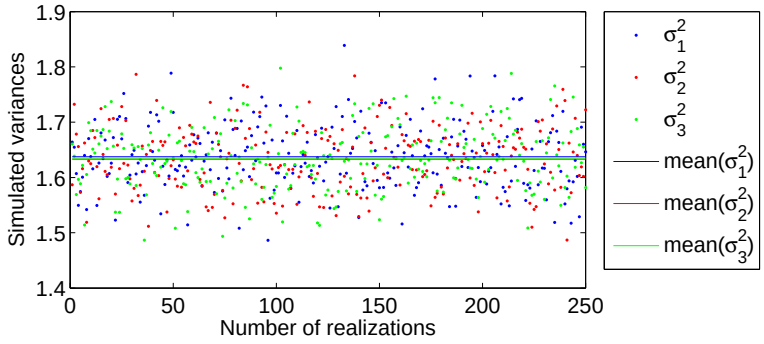



**Figure 3:** Variances of the three velocity components determined from the 250 realizations.

## 3.3 Spectrum

In Figure 4 the one-sided, one-dimensional spectrum of generated $v_1$ components is shown. The spectrum from the realization has been averaged over the $x_2$-$x_3$-plane. The theoretical spectrum given by

$$S_{11}(k_1) = \frac{18}{55}\alpha\varepsilon^{2/3}L^{5/3}\frac{1}{\left(1+L^2k_1^2\right)^{5/6}} \tag{20}$$

From the figure it is observed, that the agreement is good, but for large wave numbers the realization gives too low energy, due to aliasing errors, which was also reported by Mann (1998).

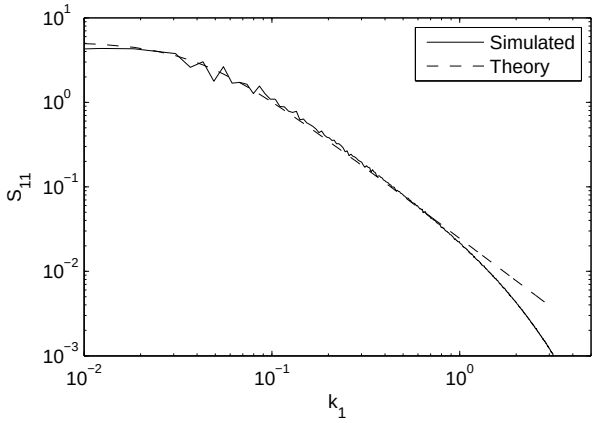



**Figure 4:** One-sided, one-dimensional spectrum of the $u_1$-components. The spectrum of the realization has been averaged over the y- and z-directions.

### 3.4 Correlation

One dimensional correlations are plotted in Figure 5. The computed correlations are compared to the theoretic functions by von Karman

$$f(r) = \frac{2}{\Gamma\left(\frac{1}{3}\right)}\left(\frac{r}{2L}\right)^{\frac{1}{3}} K_{\frac{1}{3}}\left(\frac{r}{L}\right) \tag{21}$$

$$g(r) = \frac{2}{\Gamma\left(\frac{1}{3}\right)}\left(\frac{r}{2L}\right)^{\frac{1}{3}} \left[K_{\frac{1}{3}}\left(\frac{r}{L}\right) - \frac{2}{2L} K_{\frac{2}{3}}\left(\frac{r}{L}\right)\right] \tag{22}$$

where $r$ is the separation distance and $K_a(x)$ is the modified Bessel function of second kind and order $a$. $f(r)$ describes the correlation in the direction as the velocity component (e.g. correlation of $v_2$ in the $x_2$-direction) and $g(r)$ describes the correlation in transverse directions (e.g. correlation of $v_2$ in the $x_3$-direction). Again, the agreement is good.

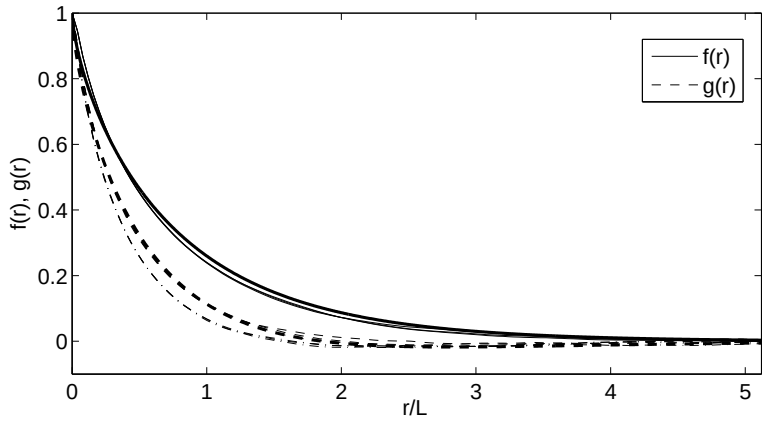



**Figure 5:** One dimensional correlation function calculated from realizations compared to the theoretic von Karman correlation functions of isotropic turbulence. Thick lines are (21) and (22) and thin are computed from realizations with the feature described in Section 2.1.

### 3.5 Divergence

The divergence of the vector field $\boldsymbol{v}$ is given by (15), and written out in components it reads

$$div(\boldsymbol{v}) = \frac{\partial v_j}{\partial x_j} = \frac{\partial v_1}{\partial x_1} + \frac{\partial v_2}{\partial x_2} + \frac{\partial v_3}{\partial x_3} \approx \frac{\Delta v_1}{\Delta x_1} + \frac{\Delta v_2}{\Delta x_2} + \frac{\Delta v_3}{\Delta x_3} \tag{23}$$

In Figure 6 the three components are compared to their sum. The derivative are approximated by their derivatives from the 2nd order difference scheme.

It can be seen that the divergence evaluated by the 2nd order central difference scheme is non-zero. By applying the algorithm from Section 2.2 the divergence is corrected to zero as shown on the bottom figure.

## 4 User's Guide

The following briefly describes the use of the developed program. The input and output is described.

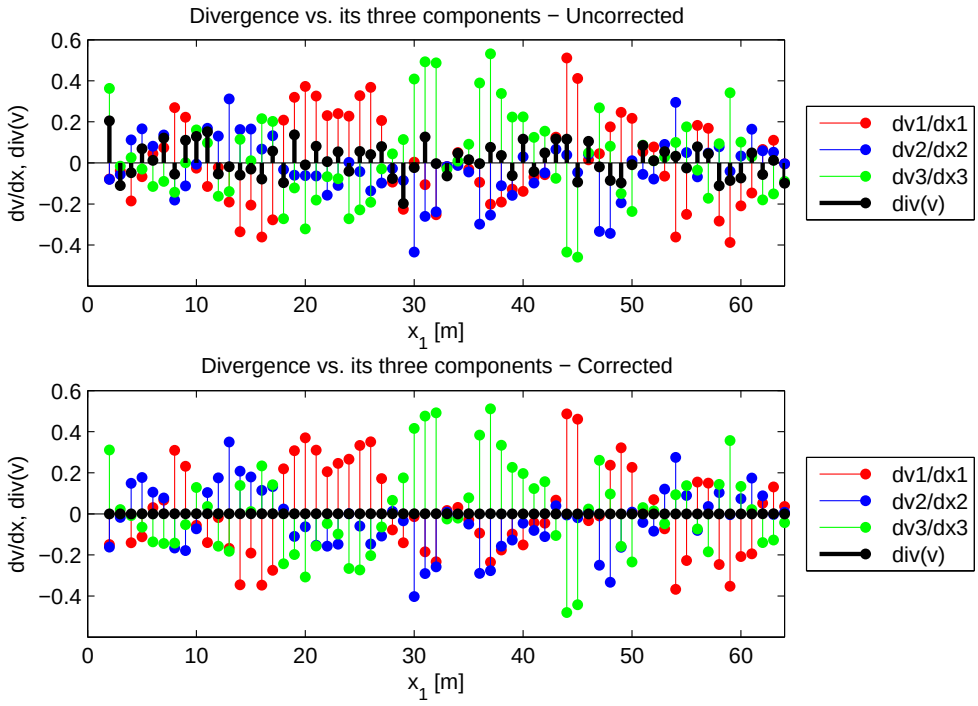



**Figure 6:** Comparison of the divergence and its three components. On the bottom figure the divergence has been corrected by the algorithm outlined in Section 2.2.

## 4.1 Input

The program reads 15 parameters from an input file. An example of such an input file is shown in Figure 7. The filename must be `input.inp`.

```
256     N1  (Variable names here are optional)
256     N2
256     N3
1.0     dx1
1.0     dx2
1.0     dx3
33.6    L
0.181   alphaeps
0.0     Gamma
-1      randseed (-1 => randseed generated from current time)
1       write velocities to file
1       print screen output
0       correct divergence
0       write variance to file
0       write correlation to file
```

**Figure 7:** Example of input file.

The first 10 lines define parameters and the remaining 5 lines are switches, that should be either 0 or 1. As stated in the file the names are optional and they are not read by the program. The order of the inputs must be exactly as in the example and no lines can be added or deleted. A blank space must separate the variables from the describing text.

The three first lines define the resolution of the domain in the three directions. These must be on the form $2^n$ where $n$ is a positive integer. The next three lines define the grid spacing in the three directions, $dx_1$, $dx_2$, $dx_3$.

$L$ is the turbulent length scale. `alphaeps` defines $\alpha\varepsilon^{2/3}$ which scales the intensity of the turbulence. $\Gamma$ (`Gamma`) is a parameter defining the degree of anisotropy of the turbulence. In Mann (1998) typical values are given.

The random seed can be set in the input file. The random seed should be an integer. If it is set to $-1$, the program will generate a random random seed and write it to an output file. The generated random seed is based on the current time.

The last five lines define switches that must be 0 or 1. If the program should write out the velocities to files, the first switch should be set to 1.
The next line defines whether or not the program should write to the screen as the execution progresses.
The third switch should be set to 1, if the program should correct the divergence of the discrete field to zero (when evaluated by the second order central difference scheme).
The second to last switch defines whether the program should determine the variances of the three velocity components to an output file.
The last switch should be set to 1, if the program should determine the correlation of the velocity field and write it to a number of files. The correlation is plotted in Figure 5.

## 4.2 Output

Depending on the parameters in the input file the program will write data to the files described here.

If the random seed is not specified (`randseed` is set to $-1$ in the input file):

- The program generates a random random seed and writes it to `randomseed.dat`

If the velocity field should be written to files:

- Velocity components are written to `u.bin`, `v.bin` and `w.bin`. The data is written in (32-bit, real*4) binary form with the 3rd index varying fastest and the 1st varying slowest. (Example of how to read the data is given below)

If the variances should be computed and written to a file:

- The computed variances will be written to the file `variance.dat`. The order is $\sigma_1$, $\sigma_2$, $\sigma_3$

If the correlation should be computed and written to files:

- One-sided, one-dimensional correlation functions are written to files `fu.dat`, `gu2.dat`, `gu3.dat`, `fv.dat`, `gv1.dat`, `gu3.dat`, `fw.dat`, `gw1.dat` and `gw2.dat`. The file names corresponds to (21) and (22). The number in the files starting with `g` is the direction and the letter is the velocity component
- Average cross-correlation coefficients of the three components is written to the file `correlation.dat`. The order is $\rho_{12}$, $\rho_{13}$, $\rho_{23}$

Except for the files containing velocity components the data is written formatted to ASCII files. The data is written in lines and if old files exist when the program is run, it will append the data from the new realization.

The velocity field can be read with Algorithm 2 (written in Fortran). The part shown is not complete. It only reads the $u$ components from `u.bin` and definition of most variables is omitted here.

**Algorithm 2**: Algorithm for reading the velocity field from files

```
real*4, allocatable :: varread(:)

allocate(varread(N1*N2*N3))
allocate(u(N1,N2,N3))
open(16,file='u.bin',form='binary')
read(16) varread
jj1=0
do j1=1,N1
   do j2=1,N2
      do j3=1,N3
         jj1=jj1+1
         u(j1,j2,j3)=varread(jj1)
      enddo
   enddo
enddo
close(16)
```

## References

Mann, J. (1998). Wind field simulation. *Prob. Engng. Mech.*, 13(4):269–282.

# 5 The Code

The code is printed in full below, and it can be downloaded from

http://vbn.aau.dk/fbspretrieve/18432733/Fortran_code_for_download.zip

A comment should be given on the FFT subroutine and how it is called. The Fourier-Stieltje components are Fourier transformed by the subroutine `fourn`. The subroutine expects a 1D array but the 3D array stored in dZ(i,:,:,:) is given as input. The trick is that Fortran stores the components in dZ in the same order as the subroutine expects them. Therefore, the calls of `fourn` will give the correct results. On some compilers the difference in dimensions gives warnings, and unfortunately on some compilers it generates errors. Yet, I have chosen this approach to save the memory otherwise required to allocate a new 1D variable that contains all the components. The argument for this choice is that memory often sets a rather strict limitation for how large fields the code can generate.

```
!  TuGen.f90
!*****************************************************************************
!  PROGRAM: TurbulenceSimulation
!  PURPOSE: Simulate turbulence field using three dimensional
!      inverse Fast Fourier transform
!  DATE:    May 2008
!  WRITTEN by:
!           Lasse Gilling, M.Sc., Ph.D.-student
!           Department of Civil Engineering
!           Aalborg University
!           Mail: lg@civil.aau.dk and/or lassegilling@hotmail.com
!*****************************************************************************
program TuGen
!*****************************************************************************
! Variables
!*****************************************************************************
implicit none
integer N1,N2,N3                ! number of points in three directions
real(8) L,alphaeps              ! Integral length scale and alpha*eps**(2./3.)
real(8) Gamma                   ! Parameters defining anisotropy
real(8) dx1,dx2,dx3             ! grid spacing in physical space
integer j1,j2,j3                ! Counters
integer jj1,jj2,jj3             ! ~Counters: Help to make program reader friendly
real(8) dk1,dk2,dk3             ! Wave number increments
real(8) k1,k2,k3                ! Wave number components
real(8) k30                     ! Initial wave number component in 3-direction
real(8) k0square, ksquare       ! Squared length of wave number vectors
real(8) E                       ! Von Karman spectral function
real(8) beta                    ! Beta in (Mann 1998)-article
real(8) B(3,3)                  ! B-matrix from "Rapid Distorsion Theory"
real(8) B1, B2, B3              ! Coefficients in B-matrix
real(8) C1,C2                   ! Parameters used for calculating B-matrix
real(8) C(3,3)                  ! C-matrix
real(8) H(3,3)                  ! H-matrix
complex*16 W(3,1)               ! Normal distributed complex random numbers
real(8) U1(3,1), U2(3,1)        ! Uniform distributed real random numbers
integer seedsize                ! Used to set the seed of the random generator
real(8) pi                      ! Ratio of diameter to circumference in circle
complex*16 S(3,1)               ! Helping variable to define elements in dZ
real(8) starttime,endtime       ! Used to determine calculation time
integer status, ios             ! Used to chech status of the ifft and read
real(8) vars(15)                ! Array used for reading input
integer randseed                ! seed for rand. gen
integer write_vel,write_screen  ! switches: write velocity, write status to screen
integer div_cor,write_var       !           correct divergence, write variances
integer write_cor               !           write correlation function f(r)
real(8) rho, rho12,rho13,rho23  ! Correlation coef.
real(8) s1_2,s2_2,s3_2          ! Variance of the the generated vel. field
real(8) tke                     ! Turbulent kinetic energy
integer time_array(8)           ! Used for generating randomseed if unspecified
complex*8, allocatable :: dZ(:,:,:,:) ! Fourier-Stieltje coefficients
real(8), allocatable :: v1(:,:,:)     ! Velocities in 1 direction
real(8), allocatable :: v2(:,:,:)     ! Velocities in 2 direction
real(8), allocatable :: v3(:,:,:)     ! Velocities in 3 direction
real(8), allocatable :: f1(:)         ! Correlation function, f(r)
real(8), allocatable :: g2(:)         ! Correlation function, g(r) , v2-comp
real(8), allocatable :: g3(:)         ! Correlation function, g(r) , v3-comp
real(4), allocatable :: VarWrite(:)   ! Write velocities to file
logical inpbool                 ! Check if the input-file exist
!*****************************************************************************
! Read input from file
```

```
!*****************************************************************************
vars=0d0
! set default control parameters. 1 for yes, 0 for no
vars(10)=-1 ! randseed
vars(11)=1  ! writevel
vars(12)=1  ! write to screen
vars(13)=0  ! divergence cor
vars(14)=0  ! compute variances and write to file
vars(15)=0  ! compute correlations and write til file
inquire(file='input.inp',exist=inpbool)
if(.not.inpbool)then
 write(*,*) 'ERROR: Cannot find input-file! Simulation aborted'
 stop
endif
open(unit=10,file='input.inp')
do j1=1,15
  read(10,*,iostat=ios) vars(j1)
  if(ios.ne.0)then
    if(j1.lt.10)then
      write(*,*) 'ERROR: Not enough data in input-file! Simulation aborted'
      stop
    endif
    exit
  endif
enddo
close(unit=10)
! Set variables
N1=vars(1)
N2=vars(2)
N3=vars(3)
dx1=vars(4)
dx2=vars(5)
dx3=vars(6)
L=vars(7)
alphaeps=vars(8)
Gamma=vars(9)
randseed=vars(10)
write_vel=vars(11)
write_screen=vars(12)
div_cor=vars(13)
write_var=vars(14)
write_cor=vars(15)
! check size of domain
call check2(N1,j1)
if(j1.ne.1)then
  print*,'Error: N1 must be on the form 2^n'
  stop
endif
call check2(N2,j1)
if(j1.ne.1)then
  print*,'Error: N2 must be on the form 2^n'
  stop
endif
call check2(N3,j1)
if(j1.ne.1)then
  print*,'Error: N3 must be on the form 2^n'
  stop
endif
!*****************************************************************************
! Initialization
!*****************************************************************************
```

```
allocate(dZ(3,N1,N2,N3))
if(write_vel.eq.1)then
  allocate(varwrite(N1*N2*N3))
endif
if(write_screen.eq.1)then
write(*,100) '+------------------------------------------------------------+'
write(*,100) '| Simulation of incompressible turbulence field has begun  |'
write(*,100) '+------------------------------------------------------------+'
endif
! Start timer
call cpu_time(starttime)
pi=4d0*datan(1d0)
! Set random seed
if(randseed.eq.-1)then
  call date_and_time (values=time_array)
  randseed=sum(time_array)*(time_array(8)+1)
  open(unit=55,file='randseed.dat',access='append')
   write(55,*) randseed
  close(55)
endif
call random_seed(SIZE=seedsize)
call random_seed(PUT=[1:seedsize]*randseed)
!*******************************************************************************
! Generate the elements of dZ (i.e. the Fourier-Stieltje coefficients)
!*******************************************************************************
dk1=2d0*pi/dx1/dfloat(N1)
dk2=2d0*pi/dx2/dfloat(N2)
dk3=2d0*pi/dx3/dfloat(N3)
do j3=1,N3
  do j2=1,N2
    do j1=1,N1
        ! ------------------------------------------------------------
        ! Generate random complex vector W
        ! ------------------------------------------------------------
        ! Generate uniformly distributed random numbers
        call random_number(U1)
        call random_number(U2)
        ! Box-Muller transformation to make normal distributed random numbers
        W=dcmplx(dsqrt(-2d0*dlog(U2))*dcos(2d0*pi*U1),&
                 dsqrt(-2d0*dlog(U2))*dsin(2d0*pi*U1))
        ! For ji=0 or Ni/2+1 the imaginary part of W must be zero
        if ((((j1.eq.1).or.(j1.eq.N1/2+1)).and.((j2.eq.1).or.(j2.eq.N2/2+1)))&
                                    .and.((j3.eq.1).or.(j3.eq.N3/2+1))) then
          W=dcmplx(dble(W),0d0)
        end if
        ! ------------------------------------------------------------
        ! Determine initial wave number vector
        ! ------------------------------------------------------------
        ! Change variable to jj that takes negative values for j.ge.N/2
        ! - i.e. shift integral range back to [-kmax;kmax[ instead of
        !   the range in iFFT [0;2*kmax[
        if(j1.ge.N1/2)then
          jj1=-N1+j1-1
        else
          jj1=j1-1
        endif
        if(j2.ge.N2/2)then
          jj2=-N2+j2-1
        else
          jj2=j2-1
        endif
```

```
if(j3.ge.N3/2)then
  jj3=-N3+j3-1
else
  jj3=j3-1
endif
! Set components of initial wave number vector
k1=jj1*dk1
k2=jj2*dk2
k3=jj3*dk3
! ------------------------------------------------------------
! Generate B-matrix to simulate anisotropy
! ------------------------------------------------------------
ksquare=k1**2+k2**2+k3**2
if (ksquare.eq.0d0) then
 ! C-matrix will be 0 and can be multiplied with anything (except NaN)
  beta=0d0
else
  beta=Gamma*(dsqrt(ksquare)*L)**(-2d0/3d0)
endif
k30=k3+beta*k1
k0square=k1**2+k2**2+k30**2
! Determine coefficients in B-matrix
if (ksquare.eq.0d0) then
  ! C-matrix will be 0 - B-matrix can be chosen arbitrarily
  B1=0d0
  B2=0d0
  B3=1d0
elseif (k1.eq.0d0) then ! use limit values for k1->0
  B1=-beta
  B2=0d0
  B3=1d0
else
  C1=beta*(k1**2)*(k0square-2*(k30**2)+beta*k1*k30) &
    /(ksquare*(k1**2+k2**2))
  C2=k2*k0square/(k1**2+k2**2)**(3d0/2d0) &
    *datan(beta*k1*((k1**2+k2**2)**0.5d0)/(k0square-k30*k1*beta))
  B1=C1-k2*C2/k1
  B2=k2*C1/k1+C2
  B3=k0square/ksquare
endif
! Assemble matrix
B(1,:)=[1d0,0d0,B1]
B(2,:)=[0d0,1d0,B2]
B(3,:)=[0d0,0d0,B3]
! ------------------------------------------------------------
! Generate C-matrix from initial wave number vector
! ------------------------------------------------------------
C(1,:)=[ 0d0,  k30, -k2 ]
C(2,:)=[-k30,  0d0,  k1 ]
C(3,:)=[ k2 , -k1 ,  0d0]
E=alphaeps*L**(5d0/3d0)*(L**2*k0square)**2/(1d0+(L**2*k0square))**(17d0/6d0)
if (k0square.eq.0d0) then
  C=0d0
else
  C=C*dsqrt(E/4d0)/pi/k0square*dsqrt(dk1*dk2*dk3)
endif
! ------------------------------------------------------------
! Create element in dZ by multiplication of matrices and vector
! ------------------------------------------------------------
S=matmul(B,matmul(C,W))
dZ(:,j1,j2,j3)=[S(1,1),S(2,1),S(3,1)]
```

```
    end do !j1
  end do !j2
end do !j3
! Enforce symmetry
do j3=1,N3
  do j2=1,N2
    do j1=1,N1
      ! ------------------------------------------------------------------------
      ! make all the symmetry conditions reqiured for the velocities to become real
      ! ------------------------------------------------------------------------
      if    (((j2.eq.1).or.(j2.eq.N2/2+1)).and.(((j3.eq.1)&
                                .or.(j3.eq.N3/2+1))).and.(j1.gt.N1/2+1))then
        dZ(:,j1,j2,j3)=conjg(dZ(:,N1-j1+2,j2,j3))
      elseif(((j1.eq.1).or.(j1.eq.N1/2+1)).and.((j3.eq.1)&
                                .or.(j3.eq.(N3/2+1))).and.(j2.gt.N2/2+1))then
        dZ(:,j1,j2,j3)=conjg(dZ(:,j1,N2-j2+2,j3))
      elseif(((j2.eq.1).or.(j2.eq.N2/2+1)).and.((j1.eq.1)&
                                .or.(j1.eq.(N1/2+1))).and.(j3.gt.N3/2+1))then
        dZ(:,j1,j2,j3)=conjg(dZ(:,j1,j2,N3-j3+2))
      elseif((j1.gt.N1/2+1).and.((j2.eq.1).or.(j2.eq.N2/2+1)))then
        dZ(:,j1,j2,j3)=conjg(dZ(:,N1-j1+2,j2,N3-j3+2))
      elseif((j1.gt.N1/2+1).and.((j3.eq.1).or.(j3.eq.N3/2+1)))then
        dZ(:,j1,j2,j3)=conjg(dZ(:,N1-j1+2,N2-j2+2,j3))
      elseif((j2.gt.N2/2+1).and.((j1.eq.1).or.(j1.eq.N1/2+1)))then
        dZ(:,j1,j2,j3)=conjg(dZ(:,j1,N2-j2+2,N3-j3+2))
      elseif((j2.gt.N2/2+1).and.((j3.eq.1).or.(j3.eq.N3/2+1)))then
        dZ(:,j1,j2,j3)=conjg(dZ(:,N1-j1+2,N2-j2+2,j3))
      elseif((j3.gt.N3/2+1).and.((j1.eq.1).or.(j1.eq.N1/2+1)))then
        dZ(:,j1,j2,j3)=conjg(dZ(:,j1,N2-j2+2,N3-j3+2))
      elseif((j3.gt.N3/2+1).and.((j2.eq.1).or.(j2.eq.N2/2+1)))then
        dZ(:,j1,j2,j3)=conjg(dZ(:,N1-j1+2,j2,N3-j3+2))
      ! ---------------------------------------------------------------
      ! The last symmetry-condition is a "mirroring" in a plane
      ! ---------------------------------------------------------------
      ! a) the x1-x2-plane
!      elseif (j3.gt.N3/2+1) then
!        dZ(:,j1,j2,j3)=conjg(dZ(:,N1-j1+2,N2-j2+2,N3-j3+2))
       !b) plane defined by x1+x2+x3=0 (Good choise => var1=var2=var3)
      elseif((dfloat(j1)/dfloat(N1)+dfloat(j2)/dfloat(N2)&
              +dfloat(j3)/dfloat(N3)-3d0/2d0).gt.0d0)then
        dZ(:,j1,j2,j3)=conjg(dZ(:,N1-j1+2,N2-j2+2,N3-j3+2))
      endif
    enddo !j1
  enddo !j2
enddo !j3
if(write_screen.eq.1)then
write(*,100) '| Fourier coefficients have been determined               |'
endif
!*****************************************************************************
! Do the iFFT
!*****************************************************************************
! Next, the Fourier-Stieltje components are Fourier Transformed. The
! subroutine expects a 1D array and dZ(i,:,:,:) is 3D. The trick is that
! Fortran stores the components in dZ in the same order as the subroutine
! expects them. Therefore, the following calls will give the correct results.
! On some compilers the difference in dimensions gives warnings, and
! unfortunately on some compilers it generates errors.
! Yet, I have chosen this approach to save the memory otherwise required to
! allocate a new 1D variable that contains all the components of dZ. The
! argument for this choice is that memory often sets a rather strict
! limitation for how large fields the code can generate.
```

```
call fourn(dZ(1,:,:,:),[N1,N2,N3],3,-1)
call fourn(dZ(2,:,:,:),[N1,N2,N3],3,-1)
call fourn(dZ(3,:,:,:),[N1,N2,N3],3,-1)
if(write_screen.eq.1)then
write(*,100) '| Velocity field has been generated                        |'
endif
!******************************************************************************
! Allocate velocities if they are used frequently
!******************************************************************************
if(div_cor+write_cor+write_var.ne.0)then
  allocate(v1(N1,N2,N3))
  allocate(v2(N1,N2,N3))
  allocate(v3(N1,N2,N3))
  v1=dble(dZ(1,:,:,:))
  v2=dble(dZ(2,:,:,:))
  v3=dble(dZ(3,:,:,:))
endif
!******************************************************************************
! Correct divergence to zero (for cds2) - Subroutine not included in this file
!******************************************************************************
if(div_cor.eq.1)then
  call div_correction(v1,v2,v3,N1,N2,N3,dx1,dx2,dx3)
 ! If divergence is corrected the veolcities are written to files here
 ! - Otherwise v1,v2,v3 may not be allocated and the velocities are written
 !   later
 ! write v1-component
  if(write_vel.eq.1)then
   open(16,file='u.bin',form='binary')
   jj1=0
   do j1=1,N1
    do j2=1,N2
     do j3=1,N3
       jj1=jj1+1
       varwrite(jj1)=dble(v1(j1,j2,j3))
     enddo
    enddo
   enddo
   write(16) varwrite
   close(16)
 ! write v2-component
   open(17,file='v.bin',form='binary')
   jj1=0
   do j1=1,N1
    do j2=1,N2
     do j3=1,N3
       jj1=jj1+1
       varwrite(jj1)=dble(v2(j1,j2,j3))
     enddo
    enddo
   enddo
   write(17) varwrite
   close(17)
 ! write v3-component
   open(18,file='w.bin',form='binary')
   jj1=0
   do j1=1,N1
    do j2=1,N2
     do j3=1,N3
       jj1=jj1+1
       varwrite(jj1)=dble(v3(j1,j2,j3))
     enddo
```

```
    enddo
   enddo
   write(18) varwrite
   close(18)
  endif
endif
!*****************************************************************************
! Write velocity field to unformatted files
!*****************************************************************************
if((write_vel.eq.1).and.(div_cor.ne.1))then
open(16,file='u.bin',form='binary')
open(17,file='v.bin',form='binary')
open(18,file='w.bin',form='binary')
do jj2=1,3
 jj1=0
 do j1=1,N1
  do j2=1,N2
   do j3=1,N3
     jj1=jj1+1
     varwrite(jj1)=dble(dZ(jj2,j1,j2,j3))
   enddo
  enddo
 enddo
 write(15+jj2) varwrite
enddo
close(16)
close(17)
close(18)
endif
!*****************************************************************************
! Do some diagnostics
!*****************************************************************************
if(write_cor.eq.1)then
! Calculate average correlation between v1 and v3 (and v1-v2, v2-v3)
  rho12=0d0
  rho13=0d0
  rho23=0d0
  do j3=1,N3
    do j2=1,N2
      call corrcoef(v1(:,j2,j3),v2(:,j2,j3),N1,rho)
      rho12=rho12+rho
      call corrcoef(v1(:,j2,j3),v3(:,j2,j3),N1,rho)
      rho13=rho13+rho
      call corrcoef(v2(:,j2,j3),v3(:,j2,j3),N1,rho)
      rho23=rho23+rho
    end do
  end do
  rho12=rho12/dfloat(N2*N3)
  rho13=rho13/dfloat(N2*N3)
  rho23=rho23/dfloat(N2*N3)
  ! write results to file
  open(50,file='correlation.dat')
  write(50,111) [rho12,rho13,rho23]
  close(50)
  ! ** Determine correlation function f(r) and g(r) in the x-direction **
  allocate(f1(N1))
  allocate(g2(N1))
  allocate(g3(N1))
  f1=0d0
  g2=0d0
  g3=0d0
```

```
do j1=1,N1
  do j2=1,N2
    call corrcoef(v1(1,j2,:),v1(j1,j2,:),N3,rho)
    f1(j1)=f1(j1)+rho/dfloat(N2)
    call corrcoef(v2(1,j2,:),v2(j1,j2,:),N3,rho)
    g2(j1)=g2(j1)+rho/dfloat(N2)
    call corrcoef(v3(1,j2,:),v3(j1,j2,:),N3,rho)
    g3(j1)=g3(j1)+rho/dfloat(N2)
  end do
end do
open(51,file='fu.dat',access='append')
write(51,111) f1
close(51)
open(51,file='gv1.dat',access='append')
write(51,111) g2
close(51)
open(51,file='gw1.dat',access='append')
write(51,111) g3
close(51)
deallocate(f1)
deallocate(g2)
deallocate(g3)
! ** Determine correlation function f(r) and g(r) in the y-direction **
allocate(f1(N2))
allocate(g2(N2))
allocate(g3(N2))
f1=0d0
g2=0d0
g3=0d0
do j2=1,N2
  do j3=1,N3
    call corrcoef(v2(:,1,j3),v2(:,j2,j3),N1,rho)
    f1(j2)=f1(j2)+rho/dfloat(N3)
    call corrcoef(v3(:,1,j3),v3(:,j2,j3),N1,rho)
    g2(j2)=g2(j2)+rho/dfloat(N3)
    call corrcoef(v1(:,1,j3),v1(:,j2,j3),N1,rho)
    g3(j2)=g3(j2)+rho/dfloat(N3)
  end do
end do
open(51,file='fv.dat',access='append')
write(51,111) f1
close(51)
open(51,file='gw2.dat',access='append')
write(51,111) g2
close(51)
open(51,file='gu2.dat',access='append')
write(51,111) g3
close(51)
deallocate(f1)
deallocate(g2)
deallocate(g3)
! ** Determine correlation function f(r) and g(r) in the z-direction **
allocate(f1(N3))
allocate(g2(N3))
allocate(g3(N3))
f1=0d0
g2=0d0
g3=0d0
do j3=1,N3
  do j1=1,N1
    call corrcoef(v3(j1,:,1),v3(j1,:,j3),N2,rho)
```

```
      f1(j3)=f1(j3)+rho/dfloat(N1)
      call corrcoef(v1(j1,:,1),v1(j1,:,j3),N2,rho)
      g2(j3)=g2(j3)+rho/dfloat(N1)
      call corrcoef(v2(j1,:,1),v2(j1,:,j3),N2,rho)
      g3(j3)=g3(j3)+rho/dfloat(N1)
    end do
  end do
  open(51,file='fw.dat',access='append')
  write(51,111) f1
  close(51)
  open(51,file='gu3.dat',access='append')
  write(51,111) g2
  close(51)
  open(51,file='gv3.dat',access='append')
  write(51,111) g3
  close(51)
endif
111 format(4096(E14.6E2))
if(write_screen.eq.1)then
if(write_vel+write_cor+write_var.ne.0)then
write(*,100) '| Data has been written to files                          |'
endif
endif
!*****************************************************************************
! Do diagnogstics and write to screen
!*****************************************************************************
if(write_var+write_screen.ne.0)then
  if(div_cor.eq.1)then
  s1_2=sum((v1(:,:,:))**2d0)/dfloat(N1*N2*N3-1)
  s2_2=sum((v2(:,:,:))**2d0)/dfloat(N1*N2*N3-1)
  s3_2=sum((v3(:,:,:))**2d0)/dfloat(N1*N2*N3-1)
  else
  s1_2=sum((real(dZ(1,:,:,:)))**2d0)/dfloat(N1*N2*N3-1)
  s2_2=sum((real(dZ(2,:,:,:)))**2d0)/dfloat(N1*N2*N3-1)
  s3_2=sum((real(dZ(3,:,:,:)))**2d0)/dfloat(N1*N2*N3-1)
  endif
endif
if(write_var.eq.1)then
open(52,file='s.dat',access='append')
write(52,111) [s1_2,s2_2,s3_2]
close(52)
endif
if(write_screen.eq.1)then
tke=0.5d0*(s1_2+s2_2+s3_2)
! some old variables are 'reused'
k1=sum(real(dZ(1,N1/2,:,:)))/dfloat(N2*N3)
k2=maxval(imag(dZ))
j1=randseed
! Determine computation time
call cpu_time(endtime)
k3=endtime-starttime
! Write to screen
write(*,100) '|-------------------------------------------------------|'
write(*,100) '| INPUT                     | CHARACTERISTICS           |'
write(*,100) '|---------------------------+---------------------------|'
write(*,101) '| N1       = ',N1,     '     | Variances of vel. comp.:  |'
write(*,102) '| N2       = ',N2,     '     | VAR1    = ',s1_2,       ' |'
write(*,102) '| N3       = ',N3,     '     | VAR2    = ',s2_2,       ' |'
write(*,103) '| dx1      = ',dx1,       ' | VAR3    = ',s3_2,       ' |'
write(*,104) '| dx2      = ',dx2,       ' | Turbulent kinetic energy: |'
write(*,103) '| dx3      = ',dx3,       ' | tke     = ',tke,        ' |'
```

```
write(*,104) '| L        = ',L  ,        ' |----------------------------|'
write(*,104) '| alphaeps = ',alphaeps,   ' | CONTROL                    |'
write(*,104) '| Gamma    = ',Gamma,      ' |----------------------------|'
write(*,105) '| randseed = ',j1,     '     | complex = ',k2,          ' |'
write(*,106) '|                           | div     = ',k1,          ' |'
write(*,100) '|-----------------------------------------------------------|'
write(*,107) '| Time elapsed: ',k3,          ' sec                        |'
write(*,100) '+-----------------------------------------------------------+'
endif
!*******************************************************************************
! Formats
!*******************************************************************************
100 format(A60)
101 format(A13,i11,A36)
102 format(A13,i11,A17,F17.6,A2)
103 format(A13,F15.6,A13,F17.6,A2)
104 format(A13,F15.6,A32)
105 format(A13,i11,A17,ES17.6,A2)
106 format(A41,ES17.6,A2)
107 format(A15,F17.6,A28)
! The end!
end program TuGen
!*******************************************************************************
! End program
!*******************************************************************************

!*******************************************************************************
! subroutine corrcoef: Determine correlation coefficient
!*******************************************************************************
subroutine corrcoef(x,y,n,rho)
! Calculate correlation coefficient between vectors x and y
! The algorithm is copy-pasted from Wikipedia but has been translated to fortran
!*******************************************************************************
! Variables
!*******************************************************************************
implicit none
! inputs:
integer n
real(8) x(n),y(n)
! output:
real(8) rho
! auxiliary
integer i
real(8) sum_sq_x, sum_sq_y, sum_coproduct, mean_x, mean_y
real(8) sweep, delta_x, delta_y, pop_sd_x, pop_sd_y, cov_x_y
!*******************************************************************************
! Calculations
!*******************************************************************************
sum_sq_x=0d0
sum_sq_y=0d0
sum_coproduct=0d0
mean_x=x(1)
mean_y=y(1)
do i=2,n
    sweep=(i-1.0)/i
    delta_x=x(i)-mean_x
    delta_y=y(i)-mean_y
    sum_sq_x=sum_sq_x+delta_x*delta_x*sweep
    sum_sq_y=sum_sq_y+delta_y*delta_y*sweep
    sum_coproduct=sum_coproduct+delta_x*delta_y*sweep
    mean_x=mean_x+delta_x/i
```

```
    mean_y=mean_y+delta_y/i
end do
pop_sd_x=sqrt(sum_sq_x/n)
pop_sd_y=sqrt(sum_sq_y/n)
cov_x_y=sum_coproduct/n
rho=cov_x_y/(pop_sd_x*pop_sd_y)
end subroutine
!*****************************************************************************
! end subroutine
!*****************************************************************************

!*****************************************************************************
! subroutine fourn: do n-dimensional fft
!*****************************************************************************
SUBROUTINE fourn(fourdata,nn,ndim,isign)
!  (C) Copr. 1986-92 Numerical Recipes Software *$3.
!     Translated to f90-format by
!     Lasse Gilling, Aalborg University.
!     April 16, 2008
!*****************************************************************************
! Variables
!*****************************************************************************
implicit none
integer isign,ndim,nn(ndim)
real(4) fourdata(*)
integer i1,i2,i2rev,i3,i3rev,ibit,idim,ifp1,ifp2,ip1,ip2,ip3
integer k1,k2,n,nprev,nrem,ntot
real tempi,tempr
real(8) theta,wi,wpi,wpr,wr,wtemp
!*****************************************************************************
! Calculations
!*****************************************************************************
ntot=1
do 11 idim=1,ndim
  ntot=ntot*nn(idim)
11 continue
nprev=1
do 18 idim=1,ndim
  n=nn(idim)
  nrem=ntot/(n*nprev)
  ip1=2*nprev
  ip2=ip1*n
  ip3=ip2*nrem
  i2rev=1
   do 14 i2=1,ip2,ip1
    if(i2.lt.i2rev)then
      do 13 i1=i2,i2+ip1-2,2
        do 12 i3=i1,ip3,ip2
          i3rev=i2rev+i3-i2
          tempr=fourdata(i3)
          tempi=fourdata(i3+1)
          fourdata(i3)=fourdata(i3rev)
          fourdata(i3+1)=fourdata(i3rev+1)
          fourdata(i3rev)=tempr
          fourdata(i3rev+1)=tempi
12      continue
13    continue
    endif
    ibit=ip2/2
1   if ((ibit.ge.ip1).and.(i2rev.gt.ibit)) then
      i2rev=i2rev-ibit
```

```
      ibit=ibit/2
    goto 1
    endif
    i2rev=i2rev+ibit
14 continue
  ifp1=ip1
2 if(ifp1.lt.ip2)then
    ifp2=2*ifp1
    theta=isign*6.28318530717959d0/(ifp2/ip1)
    wpr=-2.d0*sin(0.5d0*theta)**2
    wpi=sin(theta)
    wr=1.d0
    wi=0.d0
    do 17 i3=1,ifp1,ip1
      do 16 i1=i3,i3+ip1-2,2
        do 15 i2=i1,ip3,ifp2
          k1=i2
          k2=k1+ifp1
          tempr=sngl(wr)*fourdata(k2)-sngl(wi)*fourdata(k2+1)
          tempi=sngl(wr)*fourdata(k2+1)+sngl(wi)*fourdata(k2)
          fourdata(k2)=fourdata(k1)-tempr
          fourdata(k2+1)=fourdata(k1+1)-tempi
          fourdata(k1)=fourdata(k1)+tempr
          fourdata(k1+1)=fourdata(k1+1)+tempi
15      continue
16    continue
      wtemp=wr
      wr=wr*wpr-wi*wpi+wr
      wi=wi*wpr+wtemp*wpi+wi
17  continue
    ifp1=ifp2
  goto 2
  endif
  nprev=n*nprev
18 continue
return
end subroutine
!*****************************************************************************
! end subroutine
!*****************************************************************************

!*****************************************************************************
! Subroutine check2: Check if Nj is on the form 2^n
!*****************************************************************************
subroutine check2(Nj,flag)
implicit none
integer Nj,flag
real(8) N

N=float(Nj)
do while (N.gt.1d0)
  N=N/2d0
enddo
if(N.eq.1d0)then
  flag=1
else
  flag=0
endif
end subroutine
!*****************************************************************************
! end subroutine
```

```
!******************************************************************************

!******************************************************************************
! SUBROUTINE: div_correction, correct divergence to zero (in the cds2 scheme)
!******************************************************************************
subroutine div_correction(v1,v2,v3,N1,N2,N3,dx1,dx2,dx3)
implicit none
! input
integer N1,N2,N3
real(8) v1(N1,N2,N3),v2(N1,N2,N3),v3(N1,N2,N3)
real(8) dx1,dx2,dx3
! auxiliary
real(8) p(N1*N2*N3),divvec(N1*N2*N3),dP(N1,N2,N3)
integer m,j1,j2,j3
integer j1p,j1m,j2p,j2m,j3p,j3m
!******************************************************************************
! Remove the divergence from the velocity field
!******************************************************************************
m=0
do j3=1,N3
 do j2=1,N2
  do j1=1,N1
   if(j1.eq.1)  then; j1p=2;    j1m=N1
   elseif(j1<N1)then; j1p=j1+1; j1m=j1-1
   else;              j1m=N1-1; j1p=1
   end if
   if(j2.eq.1)  then; j2p=2;    j2m=N2
   elseif(j2<N2)then; j2p=j2+1; j2m=j2-1
   else;              j2m=N2-1; j2p=1
   end if
   if(j3.eq.1)  then; j3p=2;    j3m=N3
   elseif(j3<N3)then; j3p=j3+1; j3m=j3-1
   else;              j3m=N3-1; j3p=1
   end if
   m=m+1
   divvec(m)=(v1(j1p,j2,j3)-v1(j1m,j2,j3))/2d0/dx1 &
            +(v2(j1,j2p,j3)-v2(j1,j2m,j3))/2d0/dx2 &
            +(v3(j1,j2,j3p)-v3(j1,j2,j3m))/2d0/dx3
   end do
 end do
end do
write(*,100) ' Starting the iterative procedure to correct the divergence '
print*, '-----------------------------------'
call minres(p,divvec,n1,n2,n3,dx1,dx2,dx3)
print*, '-----------------------------------'
m=0
do j3=1,N3
 do j2=1,N2
  do j1=1,N1
   m=m+1
   dP(j1,j2,j3)=p(m)
  end do
 end do
end do
m=0
do j3=1,N3
 do j2=1,N2
  do j1=1,N1
   if(j1.eq.1)then;   j1p=2;    j1m=N1
   elseif(j1<N1)then; j1p=j1+1; j1m=j1-1
   else;              j1m=N1-1; j1p=1
```

```
   end if
   if(j2.eq.1)then;   j2p=2;    j2m=N2
   elseif(j2<N2)then; j2p=j2+1; j2m=j2-1
   else;              j2m=N2-1; j2p=1
   end if
   if(j3.eq.1)then;   j3p=2;    j3m=N3
   elseif(j3<N3)then; j3p=j3+1; j3m=j3-1
   else;              j3m=N3-1; j3p=1
   end if
   m=m+1
   v1(j1,j2,j3)=v1(j1,j2,j3)-(dP(j1p,j2,j3)-dP(j1m,j2,j3))/2d0/dx1
   v2(j1,j2,j3)=v2(j1,j2,j3)-(dP(j1,j2p,j3)-dP(j1,j2m,j3))/2d0/dx2
   v3(j1,j2,j3)=v3(j1,j2,j3)-(dP(j1,j2,j3p)-dP(j1,j2,j3m))/2d0/dx3
  end do
 end do
end do
write(*,100) ' Divergence corrected succesfully                          '
100 format(A60)
end subroutine
!*****************************************************************************
! End Subroutine
!*****************************************************************************

!*****************************************************************************
! SUBROUTINE: minres,  solve linear system by minres algorithm
!*****************************************************************************
subroutine minres(x,b,n1,n2,n3,dx1,dx2,dx3)
implicit none
!*****************************************************************************
! Variables
!*****************************************************************************
! input
integer n1, n2, n3
real(8) dx1, dx2, dx3
real(8) b(n1*n2*n3)
! output
real(8) x(n1*n2*n3)
! auxiliary
integer k
real(8) r(n1*n2*n3), p(n1*n2*n3), Ap(n1*n2*n3), Ar(n1*n2*n3)
integer M
real(8) alpha, beta, rArnew, rArold, maxtol
!*****************************************************************************
! The "matrix-vector multiplication"
!*****************************************************************************
M=n1*n2*n3
x=0d0
r=b
p=r
call Amatvec(p,n1,n2,n3,dx1,dx2,dx3,Ap)
rArold=dot_product(r,Ap)
maxtol=abs(rArold)*1d-12
do k=1,M
    alpha=rArold/dot_product(Ap,Ap)
    x=x+alpha*p
    r=r-alpha*Ap
    call Amatvec(r,n1,n2,n3,dx1,dx2,dx3,Ar)
    rArnew=dot_product(r,Ar)
    beta=rArnew/rArold
    rArold=rArnew
    Ap=Ar+beta*Ap
```

```
    p=r+beta*p
    print*, 'Iteration nr.: ', k, ' Residual: ', rArold
    if (abs(rArold)<maxtol) then
        exit
    end if
end do
100 format(A60)
end subroutine
!*****************************************************************************
! End Subroutine
!*****************************************************************************

!*****************************************************************************
! SUBROUTINE: Amatvec,  Matrix-vector-multiplication
!*****************************************************************************
subroutine Amatvec(x,n1,n2,n3,dx1,dx2,dx3,b)
implicit none
!*****************************************************************************
! Variables
!*****************************************************************************
! input
integer n1,n2,n3
real(8) dx1,dx2,dx3
real(8) x(n1*n2*n3)
! output
real(8) b(n1*n2*n3)
! auxiliary
integer i
real(8) f1,f2,f3
integer m,n1n2
!*****************************************************************************
! The "matrix-vector multiplication" b=A*x   (by using "black magic")
!*****************************************************************************
n1n2=n1*n2
M=n1n2*n3
f1=0.25d0/dx1**2
f2=0.25d0/dx2**2
f3=0.25d0/dx3**2
! (1)
b=-2d0*(f1+f2+f3)*x
! (2)
b([1:2*n1n2])=b([1:2*n1n2])+f3*x((M-2*n1n2+1):M)
! (12)
b([(2*n1n2+1):M])=b([(2*n1n2+1):M])+f3*x([1:(M-2*n1n2)])
! (3)
b([1:(M-2*n1n2)])=b([1:(M-2*n1n2)])+f3*x([(2*n1n2+1):M])
! (13)
b([(M-2*n1n2+1):M])=b([(M-2*n1n2+1):M])+f3*x([1:2*n1n2])
do i=0,(n3-1)
    ! (4)
    b([1:2*n1]+i*n1n2)=b([1:2*n1]+i*n1n2)+f2*x([(n1n2-2*n1+1):n1n2]+n1n2*i)
    ! (10)
    b([(2*n1+1):n1n2]+i*n1n2)=&
                      b([(2*n1+1):n1n2]+i*n1n2)+f2*x([1:(n1n2-2*n1)]+i*n1n2)
    ! (5)
    b([1:(n1n2-2*n1)]+i*n1n2)=&
                      b([1:(n1n2-2*n1)]+i*n1n2)+f2*x([(2*n1+1):n1n2]+i*n1n2)
    ! (11)
    b([(n1n2-2*n1+1):n1n2]+i*n1n2)=&
                      b([(n1n2-2*n1+1):n1n2]+i*n1n2)+f2*x([1:2*n1]+i*n1n2)
end do
```

```
do i=0,(n2*n3-1)
    ! (6)
    b([1:2]+i*n1)=b([1:2]+i*n1)+f1*x([(n1-1):n1]+i*n1)
    ! (8)
    b([3:n1]+i*n1)=b([3:n1]+i*n1)+f1*x([1:(n1-2)]+i*n1)
    ! (7)
    b([1:(n1-2)]+i*n1)=b([1:(n1-2)]+i*n1)+f1*x([3:n1]+i*n1)
    ! (9)
    b([(n1-1):n1]+i*n1)=b([(n1-1):n1]+i*n1)+f1*x([1:2]+i*n1)
end do
end subroutine
!*****************************************************************************
! End Subroutine
!*****************************************************************************
```