Documentation of BasicODT, a Simple ODT Code: Version 1

Alan Kerstein
Combustion Research Facility
Sandia National Laboratories
Livermore, CA 94551-0969

March 27, 2008

1 Introduction

1.1 Purpose

The purpose of the code BasicODT that is documented here is to provide an understandable introduction to ODT numerical implementation and to serve as a basis for development of other formulations. The code, written in Fortran 77, is a revision of a code originally written by Scott Wunsch.

1.2 Contents of the package

The code package consists of this document, a README file, the source code consisting of a set of .f files, a set of .dat input files, and a set of .dat output files for the case corresponding to the input files. The package is provided as six files: this document, in two formats (ODTdoc1.tex and ODTdoc1.pdf), README1 (an ascii file), and three archives: source1.tar, input1.tar, and output1.tar, where tar might be replaced by zip, depending on the archiving method. The numeral in these file and archive names anticipates future revisions that will be numbered sequentially. Discussion of these files in this document replaces the numeral by * so that the documentation is applicable to future versions without revising file names. It is recommended that revisions by others should be enumerated using a branching nomenclature. For example, a revision starting from source5.tar should be denoted source5-1.tar, and someone starting from, say, source5-3.tar should use the nomenclature source5-3-1.tar, etc.

The README file provides some documentation and there is additional documentation within the *.f files in source.tar. Any redundancy is intentional.

1.3 Method of documentation

This documentation begins with an overview of the sequence of flow advancement operations within the code. Then more detail is provided, including inputs, initialization, data gathering, and output. Names of variables, subroutines, and functions in the code are shown in boldface font.

1.4 Code structure, conventions, and compilation

The name of every source file is of the form B*.f, where B denotes 'basic.' The intention is to use a different convention in future extensions of the code so that source files unchanged from their original form can be distinguished from new or modified source files. The main program is **Bodt** in file Bodt.f. At the end of this file there is an **include** statement corresponding to each subroutine or function called by the main program or by any other subroutine or function. Therefore no makefile or make operation is needed. The command f77 [*options*] Bodt.f compiles the code.

Each source file name is the name of the subroutine or function that it contains, followed by .f.

The code is dimensional, using SI units.

1.5 Scope

The code and inputs are set up to simulate a constant-property time-developing channel flow driven by a specified pressure gradient. The particular physical model formulation that is used, and the output statistics that are gathered, are explained where their numerical implementation is described. The code is designed to run multiple realizations of this flow and gather statistics aggregated over realizations.

2 Flow geometry and mesh

The computational domain represents a lateral (wall-normal) line of sight through the channel flow, where the domain size **dom**, expressed in meters, is the channel height. The velocity component nomenclature follows the atmospheric sciences convention, in which the wall-normal direction corresponds to the vertical coordinate z, the streamwise coordinate (aligned with the imposed pressure gradient) is x, and the spanwise coordinate is y. **u**, **v**, and **w** are the streamwise, spanwise, and lateral velocity components, respectively.

The computational mesh consists of **N** uniformly spaced points located at $\mathbf{j} \times \mathbf{dom}/\mathbf{N}$, where **j** ranges from 1 to **N**. The flow is advanced with no-slip conditions applied at the z locations 0 and **dom**, which does not require advancement of the flow state at locations 0 and **N**, but location **N** is included in the mesh for future generalization to other flows. Though the mesh does not obey reflection symmetry relative to the flow midpoint, the flow advancement is statistically symmetric.

Each point is the center of a mesh cell whose properties are uniform within the cell. Fluxes are evaluated at cell boundaries which are half way between the cell centers on either side. Linear interpolation is used as needed evaluate properties at cell boundaries or fluxes at cell centers.

An adaptive mesh that is under development should lead to substantial computational cost reduction. It introduces considerable complexity so it is not used in the present code.

3 Initialization of the run and individual realizations

In **Bodt**, **parameter** statements specify values of parameters used to dimension arrays. **NVAL** is the number of mesh points allocated in arrays. The actual number **N** of mesh points, which is read from an input file, must be no larger than **NVAL**. **nstatVAL** sets the number of data-gathering intervals in data arrays. **nstat**, which is read from an input file, must not exceed this. **mVAL** sets the number of quantities that can be held in the data array **cstat**.

Subroutine **BReadOptions** reads the input file BOptions.dat. First it reads the number **nopt** of options to be read, then reads the first **nopt** values of the array **ioptions**. The remaining array values default to zero. The values in this array are option flags. Their meanings are defined in the README file. The present code has either value of option index 1 enabled, but only the value 0 is presently enabled for the other options. Option index 1 controls data output formats (see Sec. 6.3).

Subroutine **BReadPars** reads the input file BPars.dat. The first line of BPars.dat is the number of integer parameters, which corresponds to the code variable **nipars**. Each of the next **nipars** lines has one integer parameter. The next line is the number (**nrpars**) of real parameters. Each of the next **nrpars** lines has one real parameter. The integer and real parameters are read into arrays **ipars** and **rpars**, respectively. These parameters control the numerical algorithms but not the physical model specification or the flow configuration (with the possible exception of **Iv**; see the Appendix). These model and configuration parameters are read from input file BConfig.dat by subroutine **BReadConfig**, and are defined in the comments in that subroutine.

In subroutine **BReadPars**, certain parameters are set equal to default values that are overriden only if the specified number of parameters of corresponding type (integer or real) is set so that the parameters are read from BPars.dat. Parameters are defined in the README file.

The random number generator uses two seeds, **i1** and **i2**, that are set equal to one of five possible pairs of values, where the choice is specified by the input value that is assigned to **ipars**(6). Before the simulation begins, the random number generator, subroutine **Brng**, is called 100 times to subdue any artifact caused by the particular values chosen.

Subroutine **BInitRun** sets all initial velocity profiles equal to a small random noise whose purpose is explained in the Appendix. Thus all profiles are nominally but not identically zero, corresponding to the meaning of the value 0 of **ioptions**(9). The initial velocity profiles are written to files that are read during the initialization of each realization. Thus, each realization has the identical initial state. There is no obvious preference between this initialization and resampling the initial random noise for each realization. For cases that don't need a random noise, it doesn't matter whether the initial state is computed once for the run or recomputed for each realization.

Initialization needed for eddy sampling is explained in Sec. 5.2. Initialization needed for data collection is explained in Sec. 6.1.

The simulation is run within an outer loop over the number of realizations, **niter**. Each realization is initialized in subroutine **BInitIter**. This includes the initialization of time variables and of counters controlling time advancement (see Sec. 4) and data gathering (see Sec. 6.1). Also, velocity profiles **u**, **v**, and **w** are initialized by reading them from the files written by subroutine **BInitRun**.

4 Time sequence of advancement processes

4.1 Overview of flow advancement

Simulated flow advancement is implemented in the loop over the variable **iter**, which is executed **niter** times, where **niter** is the number of iterations (flow realizations). Initialization of variables for the run is implemented before this loop, and initialization of the current realization is done in **BInitIter**, called at the beginning of each pass through the loop over **iter**.

Time advancement of each realization is broken into sub-intervals for data-gathering purposes. The time duration of a realization, **tend**, is sub-divided in to **nstat** intervals of equal duration, indexed by **istat**. Thus there is a loop over **istat** within the loop over **iter**. Within the loop over **istat**, there is a loop over **itseg**, corresponding to a finer sub-division of each of the **nstat** time interval into **ntseg** sub-intervals indexed by **itseg**. The set of all sub-intervals within the realization is sequentially indexed by **itime**.

Within the inner loop, the first operation is an update of the end time, **tmark**, of the new sub-interval, based on sub-interval time duration **tend**/(**nstat** $\times$ **ntseg**). Then the current value of **itime** is updated. Then a **do while** loop advances the simulation until the time variable **t** exceeds **tmark**.

The simulation advances two time variables, **t** and **to**. **to** the physical time coordinate, but **t** is the time of occurrence of an anticipated future event, namely, the time at which the next eddy trial is scheduled to occur. (Eddy trials are discussed in Sec. 5.2.)

t advances after each occurrence of eddy sampling whether or not the sampled eddy is accepted. **t** is advanced by sampling its increment, in subroutine **BExp**, from an exponential distribution with mean **dt**, where **dt** is updated during the simulation to maintain efficient sampling (see Sec. 5.2). At the start of each flow realization, a default value of **dt** is computed in **BInitIter** by estimating the overall eddy rate based on the Kolmogorov scale being marginally resolved so that a balance of viscous transport and eddy effects is established at the mesh resolution scale. This default value is overridden if **nrpars** is 6 or larger. If this holds, then if **rpars(6)** is positive, it becomes the initial **dt** value, otherwise its absolute value multiplies the default value, so the default value is unchanged if **rpars(6)** $= -1$.

to is the cumulative elapsed time of advancement implemented in subroutine **BEqnStep** during the current flow realization. In this code, **BEqnStep** implements viscous advancement of the three velocity components, **u**, **v**, and **w**, and pressure-gradient forcing of **u** (see Sec. 5.1; wherever viscous advancement is mentioned, application of the pressure-gradient forcing is also implied). **to** is set to zero at the start of each flow realization in **BInitIter**.

to lags **t**. If an eddy sampled at nominal time **t** is accepted, it is deemed to occur instead at time **to**, which in general is earlier, but is never later, than **t**.

Advancement of **to** occurs when any of the following three conditions are met. The first condition is the acceptance and implementation of an eddy. When this occurs, it is followed by viscous advancement over a time interval **t** $-$ **to**, resulting in **to** being advanced to time **t**. The second condition is that the advancement of **t** associated with eddy sampling results in a lag **t**$-$**to** that exceeds a bound **td**, where **td** is a specified multiple of **dt** (see Sec. 5.1). The third condition is that **t** exceeds the current value of **tmark**. This condition causes an exit from the **do while** loop that implements sub-interval advancement. When this exit occurs, **to** is less than or (a rare occurrence) equal to **tmark**. Viscous advancement from time **to** to time **tmark** is implemented, resulting in **to** being advanced to time **tmark**.

These scenarios for **to** advancement govern the structure of the **do while** loop over **t**. The loop begins with viscous advancement to time **t**, implemented if the lag exceeds **td**. Next, an eddy is sampled and a decision is made whether to accept the eddy. If it is accepted, it is implemented, followed by viscous advancement from the current physical time **to** to the current value of **t**. Whether the eddy is accepted or not, a new eddy sampling time is selected.

This advancement method is more complicated and less accurate than implementing viscous advancement to time **t** prior to each eddy sampling. However, the latter would greatly increase computational cost because the mean time increment for eddy sampling, **dt**, is generally much smaller than the largest acceptable time step for viscous advancement. Viscous advancement involves the whole domain, but an eddy trial involves the eddy interval, which is typically much smaller, so viscous advancement is a much more costly operation. To avoid excessive computational cost, viscous advancement should occur over the largest time increments that accuracy and stability considerations allow. For this reason, viscous advancement is allowed to lag eddy sampling until one of the three conditions for viscous advancement is satisfied. Note that eddy acceptance is much less frequent than eddy sampling because the acceptance probability (see Sec. 5.3) is typically much less than unity. Likewise, **td** is chosen to be large relative to **dt**. The CFL time step may be smaller than **td** (see Sec. 5.1) but is typically large relative to **dt**.

When an eddy is accepted, the actual occurrence of the eddy is at **to** although its scheduled occurrence is at **t**. This procedure tends to shift eddy occurrences to earlier times than if viscous transport were advanced in synchrony with eddy trials. In general this does not have any cumulative effect such as an overall increase of the event frequency.

4.2 Sequencing of operations within the loop over sub-intervals

Sec. 4.1 provides an overview of process sequencing within the loop over sub-intervals. Here, all operations within the loop are described sequentially.

After **tmark** and **itime** are updated, the **do while** loop is entered unless **t** exceeds the new value of **tmark**. It is mathematically possible that **t** exceeds **tmark** prior to entering the **do while** loop, but this generally does not occur in cases of physical interest because it means that no eddy trials are scheduled to occur during the current sub-interval. If **t** exceeds **tmark**, the **do while** loop is bypassed and viscous transport is advanced to time **tmark** as follows.

The advancement time increment **delt** is evaluated. Conditional on positive **delt**, subroutine **BEqnStep** implements the advancement, as explained in Sec. 5.1. Before the advancement, subroutine **BSetOld** saves the state of the system in array **old** for use in data reduction (see Sec. 6.2). Specifically, this subroutine saves the state within the sub-range $[\mathbf{M}, \mathbf{M}+\mathbf{L}-1]$ of the computational mesh, so **M** is set equal to 1 and **L** is set equal to **N**-1 in order to save the complete system state. After the advancement, subroutine **BChange** forms the differences of various functions of the updated and prior (**old**) system states for use in data reduction (see Sec. 6.2). **BChange** is used to gather statistics both of viscous advancement and of eddy-induced state changes, flagged by values 2 and 0, respectively, of the index **jj**.

This completes the advancement of physical processes, and associated data gathering, to time **tmark**. Accordingly, the physical time **to** is set equal to **tmark**, signalling the completion of a pass through the loop over sub-intervals.

This pass through the loop is based on the atypical situation that the **do while** loop is bypassed. Now the more typical situation of entry into the **do while** loop is considered.

Within this loop, the time **t** of the next scheduled eddy trial is incremented after each eddy trial, causing the lag **t**−**to** between **t** and the physical time **to** to increase. To limit this lag, the **do while** loop begins with viscous advancement to time **t** if the lag exceeds the value **td**, whose determination is explained in Sec. 5.1. This viscous advancement is analogous to the advancement to time **tmark** that occurs immediately after the completion of the **do while** loop (see the preceding description).

The next operation in the **do while** loop is eddy sampling, performed within subroutine **BSampleEddy**. This subroutine determines the spatial range $[\mathbf{M}, \mathbf{M}+\mathbf{L}-1]$ of the candidate eddy (see Sec. 5.2) and its acceptance probability **pp** (see Sec. 5.3). **pp** is compared to a random number **rannum**, sampled from the uniform distribution over $[0, 1]$ by the subroutine **Brng** (which uses and updates seeds **i1** and **i2**), to decide whether the candidate eddy is accepted (which occurs if **pp** exceeds **rannum**). If the eddy is accepted, the eddy is implemented in the section of code in **Bodt.f** that is conditional on **pp** exceeding **rannum**. Details of eddy implementation in this section of code are described in Sec. 5.4.

Following the section of code conditional on eddy acceptance, the time **t** of the next scheduled eddy trial is updated. It is equal to the time of the eddy trial just completed plus a time increment that is sampled (in subroutine **BExp**) from the exponential distribution with mean value **dt**, where **dt** is the current value of the mean time increment between eddy trials. The final operation in the loop over sub-intervals is adjustment of the value of **dt** in subroutine **BRaisedt**, as explained in Sec. 5.2.

5 Details of flow advancement

5.1 Viscous advancement

Section 4.2 describes the circumstances under which viscous advancement occurs. Subroutine **BEqnStep** sets up the time sequencing of calls of subroutine **BAdv**, in which viscous advancement is implemented. The total advancement time increment within **BEqnStep** is **delt**. The present code sub-divides this time increment into smaller time increments that satisfy the CFL condition for explicit time advancement, which is implemented in **BAdv**. The code is written this way to accommodate other advancement schemes, such as implicit advancement, with minimal modification. An implicit scheme might need no sub-division of the time increment **delt** if this increment were controlled, by assignment of the value of **td**, so as to obtain sufficient accuracy.

In **BEqnStep**, the CFL time **tcfl** is computed as 0.5 times the square of the mesh spacing **dom/N** divided by the viscosity **visc**. The variable **tfrac**, which is set equal to the input quantity **rpars(5)**, determines the desired upper bound on time step that is used for viscous advancement. The integer variable **irat** is equated to $\mathbf{delt}/(\mathbf{tfrac} \times \mathbf{tcfl})$, resulting in truncation of this double-precision quantity to an integer value. The time step for viscous advancement is then taken to be $\mathbf{et} = \mathbf{delt}/(\mathbf{irat} + 1)$, which cannot exceed the desired upper bound and allows advancement over a total time increment **delt** in **irat** + 1 equal time steps. The advancement over this number of time steps is implemented in a loop over index **i**.

Within this loop, subroutine **BStats** gathers data, as explained in Sec. 6.1. Then, calls of subroutine **BAdv** advance the **u**, **v**, and **w** velocity profiles, respectively, forward in time by the amount **et**.

This time advancement includes not only viscous advancement, but additional terms of the evolution equations, excluding the advective term that is modeled by eddy events. For the present application, the only additional term is the imposed mean pressure gradient in the streamwise direction. Thus, subroutine **BAdv** advances the equation

$$dr/dt = \nu d^2r/dz^2 + G, \tag{1}$$

where r is a specified input array, ν is the kinematic viscosity and G is specified by the dummy argument **force** of subroutine **BAdv**. If r is the streamwise velocity **u** in the subroutine call, then G is assigned to be the variable **pgrad** in the subroutine call, corresponding to the physical quantity $-(1/\rho)dP/dx$, where P is pressure and x is the streamwise coordinate. (**pgrad** is read from input file BConfig.dat by subroutine **BReadConfig**.) Note that it is not necessary to specify the density ρ or the pressure gradient dP/dx individually. If r is the lateral (wall-normal) velocity **w** or the spanwise velocity **v**, then $G = 0$, which is assigned by the argument **zero** in the subroutine call.

In other contexts, G can represent other forcing mechanisms such as the Coriolis force, which is important in atmospheric and other rotating flows, and a source or sink in cases for which r is a non-conserved scalar such as a reacting chemical species. In many cases, new or modified physics can be introduced by changing the form of G or by introducing additional property fields r without changing the overall modeling framework or code structure.

In **BAdv**, Eq. 1 is advanced a time increment **et** (the value assigned to the dummy variable **tstep** in the subroutine call) by forming fluxes **f** that are −**visc** times the first-difference approximation of dr/dz. The first difference of the fluxes then approximates d^2r/dz^2. The no-slip conditions imply that r remains fixed at the top and bottom of the physical domain, so r is not advanced at location **N**.

Viscous advancement can readily be modified to accommodate periodic boundary conditions. However, periodic boundary conditions also require modified implementation of eddy events, which needs some care, but is not a major effort. Data gathering and reduction is also affected due to the role of eddy effects in determining advective fluxes (Sec. 6.2).

In subroutine **BInitIter**, **td** is assigned to be a specified multiple **tdfac** of the mean time **dt** between eddy trials, where **tdfac** is a assigned to be the input quantity **rpars(4)**. **td** is re-evaluated in the same way in subroutine **BRaisedt** whether or not **dt** is changed within the subroutine but **td** is not re-evaluated in **BLowerdt**.

5.2 Eddy sampling

A key part of the turbulent flow representation in ODT is the specification of an eddy rate distribution (see the Appendix). On the discrete mesh, this implies a rate of eddy occurrences for each possible choice of the integer variables $\mathbf{M}$ and $\mathbf{L}$ that specify a particular eddy.

The eddy rate for given $\mathbf{M}$ and $\mathbf{L}$ depends on the flow state specified by velocity profiles $\mathbf{u}$, $\mathbf{v}$, and $\mathbf{w}$ within the spatial range $[\mathbf{M}, \mathbf{M}+\mathbf{L}-1]$ affected by the eddy. This implies a need to recompute the rate for each $\mathbf{M}$ and $\mathbf{L}$ value (there are order $\mathbf{N}^2$ of these values) whenever viscous advancement changes the flow state. This procedure is unaffordable, but fortunately there is a mathematically more elaborate but computationally much more efficient method for doing the sampling, called 'thinning' [3].

To introduce the method, suppose that there is only one type of eddy and that its rate of occurrence $\lambda(t)$ is an unknown function of t (but can be evaluated at given t when needed), except that it is known that it is always less than some value λ^*. Assuming that occurrences are statistically independent in time, a valid statistical sample of occurrences is generated as follows. Generate random occurrences of Poisson events with uniform (in time) rate λ^*. For an occurrence at given time $\hat{t}$, evaluate $\lambda(\hat{t})$ and decide the acceptance or rejection of $\hat{t}$ as a time of occurrence of the process of interest probabilistically. Namely, it is accepted with probability $\lambda(\hat{t})/[\lambda^* h(\mathbf{M}, \mathbf{L})]$, otherwise rejected. Note that this requires only a finite number of λ evaluations during any finite time interval, in contrast to the nominally continuous-in-time evaluation that would otherwise be required for an exact sampling procedure. The assumption that λ^* is uniform in time can be relaxed provided that $\lambda^* > \lambda$ for all t, and the eddy sampling procedure exploits this flexibility in order to gain efficiency.

The problem of interest is more complicated because it involves a two-parameter family of possible events. For this application, the method is generalized as follows. Instead of using a single majorant λ^*, each member of the family of events is majorized using the function $\lambda^* h(\mathbf{M}, \mathbf{L})$, where h is normalized as a joint probability distribution of $\mathbf{M}$ and $\mathbf{L}$. This function can then be sampled for eddy occurrences by first sampling the occurrence time from the Poisson process with rate λ^* and then sampling the eddy parameters from $h(\mathbf{M}, \mathbf{L})$. This provides the information for the computation of λ for the selected eddy parameters at time $\hat{t}$ based on the theory explained in the Appendix. The sampled eddy is then accepted with probability $\lambda/[\lambda^* h(\mathbf{M}, \mathbf{L})]$. The construction of and sampling from the function h is considered first, followed by a discussion of sampling of the time $\hat{t}$ of occurrence of a trial eddy.

The choice of the function h affects computational efficiency but not the statistical ensemble of realizations. In the present code, h is chosen to be a separable function $h(\mathbf{M}, \mathbf{L}) = f(\mathbf{L})g(\mathbf{M})$. The assumed forms and numerical evaluation of the functions f and g (which are, individually, probability distributions) are discussed in the Appendix.

The function $f(\mathbf{L})$ is evaluated once, at the beginning of the run, in subroutine **BLenProb**. It depends on several parameters that are read from input file BPars.dat during run initialization. The evaluation of this function yields the output **PL**, which is an array of partial sums (the cumulative distribution) of f, a convenient form for sampling from the distribution. Also provided by subroutine **BLenProb** are variables **Co** and **Cv** that are used to find a good starting point for the procedure that samples from the distribution. It is convenient to construct and sample from **PL** defined as a function of the integer **I** where $\mathbf{L} = 3 \times \mathbf{I}$.

Subroutine **BSampleEddy** calls subroutine **BLength**, in which a value of **I**, denoted **L3** in the calling routine **BSampleEddy**, is sampled. In **BLength**, $\mathbf{nn} = [\mathbf{x}] + 1$ is an initial guess of **I**, where [ ] denotes integer part and **x** is based on an analytical approximation of the discrete function **PL** (see the Appendix). **I** is sampled from **PL** by first picking a random number, **rannum**, between 0 and 1 (uniformly distributed), next evaluating the initial guess for **nn**, and then increasing **nn** in integer steps until **PL**(**nn**) is no less than **rannum**. Then **nn** is decreased in integer steps until $\mathbf{PL}(\mathbf{nn} - 1)$ is no greater than **rannum**. By the definition of the cumulative distribution, this procedure selects the value **nn** with probability $\mathbf{PL}(\mathbf{nn}) - \mathbf{PL}(\mathbf{nn} - 1)$, as desired. Roundoff could cause branching to statement 20 in **BLength** even if $\mathbf{PL}(\mathbf{nn} - 1)$ is nominally zero, so the subroutine concludes with a statement that prevents **I** from falling below its specified minimum value **Io**, which is an input parameter (see the Appendix).

After **L3** and $\mathbf{L} = 3 \times \mathbf{L3}$ are evaluated in subroutine **BSampleEddy**, **M** is sampled from the uniform distribution over integers 1 through $\mathbf{N} - \mathbf{L}$. This choice of the range of possible **M** values excludes the possibility that point **N** is within the eddy range. Point **N** corresponds to one of the channel walls. Like the wall at nominal location zero, which is not included in spatial arrays, it is not subject to eddy events. Strictly speaking, no inconsistency would result from including zero and **N** in eddies because the current numerical implementation of eddies has no effect on the first and last points, **M** and $\mathbf{M} + \mathbf{L} - 1$, of the eddy range. Either choice is valid; the distinction between them becomes negligible as the mesh is refined to obtain mesh-independent statistical results.

In the sampling of **M**, the **min** function guards against the possible occurrence of **rannum** so close to unity that the **int** function truncates its argument to $\mathbf{N} - \mathbf{L}$, which is unlikely but not impossible. Note that truncation to zero results in the selection $\mathbf{M} = 1$.

With eddy parameters thus selected, eddy acceptance or rejection can be decided, starting with the use of subroutine **BProb** to evaluate the acceptance probability **pp** in subroutine **BSampleEddy**. Additional details are explained in Sec. 5.3 and the Appendix.

The sampling of the time $\hat{t}$ is revisited. In Sec. 4.2, it is mentioned that the time increment from a given eddy trial to the next is sampled from the exponential time distribution with mean **dt**. This is equivalent to sampling a time increment of a Poisson process with mean rate $\lambda^* = 1/\mathbf{dt}$, thus establishing the connection between the sampling of **dt** and the mathematical statement of the thinning algorithm.

As noted, a requirement of the thinning algorithm is that the rate at which trial eddies are sampled for any values of **M** and **L** must majorize (exceed) the actual rate of occurrence, which is determined only when $\lambda(\hat{t})$ is computed. To enforce this requirement without being overly conservative, the magnitude of **dt** is adjusted as the flow evolves.

One type of adjustment occurs in subroutine **BLowerdt**, called from **BSampleEddy**. If the computed acceptance probability **pp** exceeds a value **pmax** corresponding to the input quantity **rpars(1)**, then **dt** is multiplied by the factor **pmax/pp**, which reduces **pp** by this factor because the acceptance probability is the desired sampling rate divided by the actual rate, which is inversely proportional to **dt** (see the Appendix).

The validity of this procedure relies on a particular interpretation of the relationship between trial eddy occurrences and the sampling process. Eddy sampling at time **t** is deemed to be associated with the probability of eddy occurrence during $[\mathbf{t}, \mathbf{t} + \mathbf{dt}]$ (forward in time) rather than $[\mathbf{t} - \mathbf{dt}, \mathbf{t}]$ (backward in time), consistent with the requirement that eddy sampling leads rather than lags the physical time **to**. Therefore **dt** can be changed until the random trial determining eddy acceptance is performed. A more subtle question is whether the exceedance of **pmax** is a manifestation of undersampling of particular eddies in violation of the requirements of the thinning algorithm. In practice, **pmax** is assigned a value well below unity to avoid any significant effect of this scenario on ensemble statistics. Also, cases are run with different values of **pmax** to check sensitivity to this parameter, analogous to a test of sensitivity of a deterministic marching scheme to the time step.

Another type of adjustment occurs in subroutine **BRaisedt**, called from **Bodt** immediately after **t** is updated. The adjustment is attempted only if the counter **ii** is an exact multiple of the integer **iwait** whose value is set by the input quantity **ipars(2)**. In subroutine **BInitIter**, **ii** is initialized to zero at the beginning of each simulated realization. **ii** is incremented by unity at the beginning of subroutine **BSampleEddy**, so it counts all eddy trials, including rejected eddies.

The adjustment is based on parameters **pa** and **Np** that are initialized to zero in **BInitIter**, incremented in **BLowerdt**, and reinitialized to zero after the attempted **dt** adjustment in **BRaisedt**. In **BLowerdt**, if the computed eddy acceptance probability **pp** is nonzero, the counter **Np** is incremented by unity and **pa** is incremented by the value of **pp**. In **BRaisedt**, the sum **pa** is converted into an average by dividing by **Np**. If this average is smaller than a minimum value **pmin** specified by the input quantity **rpars(2)**, then **dt** is increased by the smaller of the two factors **dtfac** and **pmin/pa**, where the maximum increase factor **dtfac** is specified by the input quantity **rpars(3)**.

5.3 Eddy acceptance

The computation of the acceptance probability **pp** in subroutine **BProb**, called in **BSampleEddy**, introduces much of the physical content of the model. For this reason, the explanation of computational details accompanies the theoretical discussion in the Appendix. Here, two features are noted. First, the theoretical expression for **pp** can yield a negative acceptance probability, which guarantees rejection of the trial eddy. In this case **pp** is set equal to zero. Second, the quantities **uK**, **vK**, and **wK** that are computed in **BProb** for use in computing **pp** are also needed for implementation of an accepted eddy, so they are passed up through the calling chain to **Bodt**, where they are available for later use in subroutine **BEddy**.

5.4 Eddy implementation

If the comparison of **rannum** to **pp** indicates eddy acceptance, then the section of code that is conditional on eddy acceptance is entered (see Sec. 4.2). Before eddy implementation, statistics based on the current flow state within the eddy interval $[\mathbf{M}, \mathbf{M}+\mathbf{L}-1]$ are saved in array **old** by subroutine **BSetOld**. After eddy implementation, these statistics and the updated flow state are used by subroutine **BChange** to increment the data-gathering array **edstat** (see Sec. 6.1).

Eddy implementation occurs in subroutine **BEddy**. This involves two types of operations. One is application, in subroutine **BTriplet**, of the permutation representing the discrete triplet map applied to points in the range $[\mathbf{M}, \mathbf{M}+\mathbf{L}-1]$. The other is addition of kernels to velocity profiles in subroutine **Baddk**.

Subroutine **BTriplet** permutes the points of a size-**N** array **s** as follows. It leaves points 1 through $\mathbf{M}-1$ and $\mathbf{M}+\mathbf{L}$ through **N** unaffected, where **L** is three times an integer **Lo**. Starting from point **M**, every third point of array **s** is written to scratch array **x** consecutively until **x** receives **Lo** points by this procedure. Then **x** receives **Lo** additional points, starting from point $\mathbf{M}+\mathbf{L}-2$ of array **s** and moving downward in that array, again with a stride of three. Again, these points are written consecutively to **x**, starting at point $\mathbf{Lo}+1$ of that array. The next set of **Lo** points of array **x** receives points written from points of array **s**, starting at point $\mathbf{M}+2$ of that array and proceeding upward with a stride of three. Finally, points 1 through **L** of scratch array **x** are copied to points **M** through $\mathbf{M}+\mathbf{L}-1$ of array **s**, completing the permutation of that array. The permutation leaves points **M** and $\mathbf{M}+\mathbf{L}-1$ of array **s** unaffected, so strictly speaking there is no need to include these points in the algorithm. In subroutine **BEddy**, **BTriplet** is called three times in order to apply this permutation to the three velocity profiles **u**, **v**, and **w** that specify the flow state.

One way to represent the discrete triplet map is to form the array of map-induced displacements of the points of array **s**. Take the array of indices of points of the mapped array **s**, which is simply the list of consecutive positive integers starting from 1, and subtract from each index the index of the point mapped to that location by a particular discrete triplet map. The resulting array is the map-induced displacement, or index change, of each point, arranging according to post-map location. This array is called the kernel of the map.

Subroutine **BAddK** adds a kernel, times a quantity **c**, to a dummy array **r**. The kernel corresponds to the triplet map with the usual definitions of **M** and **L** = 3**Lo**. It is convenient to loop over an index **j** running from 1 to **Lo**, where for given **j**, the kernel is evaluated at the three locations **M**+**j**−1, **M**+**j**+**Lo**−1, and **M**+**j**+2**Lo**−1, which are denoted by the variables **j1**, **j2**, and **j3** respectively. The kernel values corresponding to these locations, denoted **y1**, **y2**, and **y3** respectively, are evaluated in the loop. The respective points **j1**, **j2**, and **j3** of array **r** are incremented by these values, multiplied by **c**. The theory and numerical implementation on which the kernel specification and the computation of **c** (in subroutine **BEddy**) are based are described in the Appendix. In **BEddy**, subroutine **BAddK** is used to add the kernel times values **cu**, **cv**, and **cw** to velocity profiles **u**, **v**, and **w** respectively.

After eddy implementation, viscous advancement is implemented, as explained in Sec. 5.1, from the physical time **to** to the nominal time **t** of eddy acceptance. Because eddy implementation precedes this advancement, it is deemed to have occurred, for data gathering purposes, at the physical time **to** (see Sec. 4.1). After viscous advancement, **to** is updated to the value **t**.

6 Data processing

6.1 Data gathering

The code is set up to gather all needed data and partially reduce it on the fly. Data reduction is completed after all the needed data has been gathered. This data processing is done in the code rather than saving raw data for future postprocessing because this procedure is more efficient and also because the code then demonstrates all aspects of ODT numerical implementation.

Several forms of data gathering are done in the code. Let $s(t)$ denote either an evolved array or some function of these arrays, here suppressing the spatial argument. One form of data gathering is summation, over simulated flow realizations, of the value of s at a fixed time (here referring to physical time as defined in Sec. 4.1). This is called an i-sum (sum over iterations). It is accumulated in subroutine **BSeries**. Another form of data gathering is also a sum over realizations, but within each realization accumulates the time integral of s over some time interval. Here this is called a t-sum (time integration). It is implemented in subroutine **BStats**. The last form of data gathering is also accumulated over realizations and over some time interval within each realization, but the accumulated quantity is the change in the value of $s(t)$ that results from a particular type of operation within the code. Here this is called a c-sum (change accumulation). It is implemented in subroutine **BChange**.

All elements of all data gathering arrays are initialized to zero in subroutine **BInitStats** prior to the loop over simulated flow realizations. **parameter** statements are used in **Bodt** to set some of the dimensions of these arrays for ease of modification. i-sums are accumulated in array **umoms**, t-sums in array **cstat**, and c-sums in array **edstat**, with **old** serving as an auxiliary array aiding the accumulation of c-sums.

The accumulation of i-sums is implemented in subroutine **BSeries**, which is called in **Bodt** at the end of each sub-interval. For each sub-interval, indexed by **itime**, the sum over realizations of the velocity component **u**, evaluated at location index **ictr** equal to the integer part of $0.5 \times \mathbf{N}$, is accumulated. The analogous sum of $\mathbf{u}^2$ is also accumulated.

The accumulation of t-sums is implemented in subroutine **BStats**, which is called in **BEqnStep** prior to each set of calls of subroutine **BAdv** that advance the flow state by a time increment **et** (see Sec. 5.1). In **BStats**, elements of the array **cstat** are incremented by **tstep** times each quantity of interest, where **tstep** is a dummy time step assigned to be **et** in the call to **BStats**.

The procedure is as follows. Each quantity of interest is a spatial profile of some function of the state variables. The elements of the dummy array **term** are set equal to **tstep** times the values of this function. Then **term**, **cstat**, and array dimensions are passed to subroutine **BAddTerm**, where the t-sum array **cstat** is incremented.

The index **i** that is passed to **BAddTerm** labels the quantity that is being accumulated in an element of **cstat** whose second argument is **i**. (The first argument **j** is spatial location index and the third argument **istat** is the index of the averaging period.) For $\mathbf{i} = 1$, the time increment **tstep** is accumulated (which is the same for all **j**, a redundant but convenient procedure). After that, in order of increasing **i**, the quantities that are accumulated (times **tstep**) are **u**, **v**, **w**, $\mathbf{u}^2$, $\mathbf{v}^2$, $\mathbf{w}^2$, δ_u^2, δ_v^2, and δ_w^2, where δ^2 denotes the square of the first difference of the subscripted quantity. Evaluation of δ^2 for $\mathbf{j} = 1$ takes into account that $\mathbf{j} = 0$ is a wall location, where all state variables are zero (hence this location is omitted from state and data arrays).

c-sums are accumulated by saving a portion of the flow state before an operation, performing the operation, and then forming differences between functions of the new and old flow state. In subroutine **BSetOld**, the old flow state $(\mathbf{u}, \mathbf{v}, \mathbf{w})$ at locations **M** through $\mathbf{M} + \mathbf{L} - 1$ is saved in array **old**. When gathering eddy statistics, **M** and **L** are the eddy parameters, but when gathering viscous advancement statistics they are set equal to 1 and $\mathbf{N} - 1$, respectively so that all non-null changes are captured. The same considerations apply to the arguments **M** and **L** of subroutine **BChange**, in which the differences are formed. The argument **jj** of this subroutine is assigned the value 0 to gather eddy statistics or 2 to gather viscous advancement statistics.

In either case, data array **edstat** is incremented. The first index of **edstat** denotes the spatial location and the fourth index denotes the data averaging interval. The second index of **edstat** determines the quantity that is gathered. If its value is 1 or 2, then the change of the first or second moment of each velocity component (where the third index of **edstat**, ranging from 1 to 3, is the the component label) is gathered. Values 3 and 4 have the same meaning as 1 and 2, respectively, except that 1 and 2 correspond to eddy statistics ($\mathbf{jj} = 0$) and 3 and 4 correspond to viscous advancement statistics ($\mathbf{jj} = 2$).

For $\mathbf{jj} = 0$, eddy size histograms are also incremented. Successive size bins (first index of **edstat**) correspond to 1/3 the number of cells in the eddy. The second index distinguishes two histograms that are formed. Index value 2 corresponds to a histogram of eddies smaller than their distance from the wall (based on closest approach) and index value 1 corresponds to a histogram of all other eddies. The value 4 of the third index of **edstat** labels the sub-array in which the histograms are accumulated.

6.2 Data reduction

Reduction of the data in the i-sum array **umoms** is performed in subroutine **BWriteSeries**, which is called in **Bodt** after the completion of all simulated realizations. In **BWriteSeries**, each array element is divided by the number **niter** of simulated realizations to convert sums to averages over realizations, and the square of the average of **u** is subtracted from the average of $\mathbf{u}^2$ to form the variance of **u** in array **temp**. The physical time corresponding to each result is evaluated as the elapsed time **tend/itime** of each subinterval times the sub-interval index of the result, and is recorded in array **time**. Note that when subroutine **BWriteSeries** is called, **itime** is equal to the total number of sub-intervals during the realization.

Reduction of the data in the t-sum array **cstat** and the c-sum array **edstat** is performed in subroutine **BSnap**, which is called at the end of each data-gathering period during the last simulated realization (when **iter** has the value **niter**). In **BSnap**, each of the t-summed quantities (except the total elapsed time, corresponding to **cstat(*,1,*)**, where the entries corresponding to all possible values of the starred indices are identical) is divided by the elapsed time to obtain time averages, which are stored in array **cavg**. The c-summed quantities are likewise divided by the elapsed time to obtain c-averages **eavg** that are interpreted as rates of change of the summed quantities, where the rates are attributed to the particular process, viscous advancement or eddy implementation, whose effects were summed. In obtaining averages from sums, note that each summation extends over the time interval of the data-gathering period (index **istat**), aggregated over all the simulated realizations, where the number of realizations is **niter**. Also note that **edstat** is divided by the elapsed time only for values 1 through 3 of its third argument, **k**, because $\mathbf{k} = 4$ labels a sub-array containing eddy size histograms (see Sec. 6.1). These histograms are written to output (see Sec. 6.3) without further processing.

cavg includes time averages needed to compute velocity moments. Without further processing, **cavg(*,2)** is the profile of mean streamwise velocity. Averages of other velocity components are identically zero so they are not written to output (see Sec. 6.3). The variance of each component is computed as the average of the square minus the square of the average, although the subtraction is superfluous for the **v** and **w** components. Moreover, the **v** and **w** variances, both of which are computed, are redundant because the statistics of these components are identical for the model formulation and flow configuration used in this code.

eavg(j,1,1) is the rate of change of **u** in cell **j** caused by eddy events. Therefore, the cell size (denoted **width** in **BSnap**) times the sum of **eavg(j,1,1)** for all **j** exceeding some value j_0 is the eddy-induced flux of **u** across the boundary between cells j_0 and $j_0 + 1$. Because eddy events are the sole mechanism of wall-normal advective motion in the present formulation, this quantity is the wall-normal advective flux of **u**. On this basis, the wall-normal profile of this quantity is computed. This definition takes into account that kernel addition as well as the triplet map induces momentum transfer.

The wall-normal velocity **w** is not involved in the evaluation of wall-normal advective fluxes because advection along the ODT domain is not governed by **w**. **w** (and other velocity components) have only an indirect influence on wall-normal advection through their role in eddy selection (see the Appendix). This point applies also to the computation of Reynolds stresses, which is explained later in this section.

The turbulent kinetic energy (TKE) budget consists of four terms, production **p**, dissipation **d**, advective transport **ta**, and viscous transport **tv**. Only wall-normal transport is represented in the budget, which is specific to statistically steady (fully developed) flow. The two types of transport are attributed to eddy-induced changes and viscous advancement, respectively. Production is a purely advective process, and hence is associated with eddy events. Dissipation is a purely viscous process.

A formal expression for the budgets of velocity component variances is derived in the Appendix of [2]. It simplifies somewhat when the variance budgets are summed and divided by 2 to form the TKE budget. The terms of the TKE budget are discussed individually.

The theoretical expression for the viscous transport term is the sum over i of $\frac{\nu}{2}\frac{\partial^2}{\partial z^2}\langle v_i'^2\rangle$ where v_i denotes a velocity component and prime denotes deviation from mean value. To evaluate this, $\langle v_i'^2\rangle$ is expressed as $\langle v_i^2\rangle - \langle v_i\rangle^2$, involving moments of v_i that are evaluated as t-averages in array **cavg**, as noted in the prior discussion of the evaluation of the variance $\langle v_i'^2\rangle$. The viscous transport term is evaluated by summing component variances, taking the second difference, dividing by the square of the cell width to convert the second differences to as second derivative, and multiplying by $\nu/2$. The variable **fluxfac2** contains most of the needed factors.

In 3D flow, velocity gradients in all three directions contribute to viscous dissipation even in flows that are statistically homogeneous in one or more directions. However, viscous advancement in ODT occurs only in the z (wall-normal) direction, so dissipation involves gradients only in this direction. The dissipation term is $\nu\langle\frac{\partial v_i'}{\partial z}\frac{\partial v_i'}{\partial z}\rangle$, summed over i. For evaluation, v_i' can be expressed as $v_i - \langle v_i\rangle$, giving $\nu\langle\frac{\partial v_i}{\partial z}\frac{\partial v_i}{\partial z}\rangle - \nu(\partial\langle v_i\rangle/\partial z)^2$. Evaluation on this basis is implemented in **BSnap** using t-averages from the array **cavg**, but farther down in the code this evaluation is superseded by an alternate evaluation that is explained shortly.

The evaluation using t-averages is unsuitable because the numerical advancement in subroutine **BAdv** allows the pressure-gradient forcing, which is applied to the **u** component, to make a small contribution to TKE in addition to its contribution to mean kinetic energy. (Exact numerical implementation of the model would preclude any TKE contribution by the pressure-gradient forcing.) Therefore the theoretical expression for TKE dissipation does not capture precisely the TKE change induced by the numerical procedure. The discrepancy is not significant with respect to physical implications, such as comparison to measurements, but it prevents precise balance of the TKE budget. It is useful to obtain precise balance as a check of both the simulation and the data reduction, so the following alternate evaluation of TKE dissipation is used.

TKE dissipation at a given location **j** is expressed as the mean rate of reduction of total kinetic energy minus the mean rate of reduction of mean kinetic energy plus the viscous transport contribution to the rate of TKE change. The computation of the latter has been explained. The contribution of velocity component **i** to the mean rate of reduction of total kinetic energy is -0.5 times **eavg(j,4,i)**. This is the actual rate of reduction based on the change induced by the numerical advancement, rather than a theoretical expression for the change. The negative of the mean rate of reduction of mean kinetic energy is the mean velocity, **cavg(j,i+1)**, times the rate of change of the mean velocity, **eavg(j,3,i)**. The alternate evaluation of TKE dissipation combines the contributing terms, with summation over components **i**.

The computation of viscous transport using t-averaged quantities to evaluate the theoretical formula is subject to the same numerical error as the use of the theoretical formula for dissipation. By construction, the alternate evaluation of dissipation is subject to an error that is equal and opposite to the viscous transport numerical error such that balance of the TKE budget is obtained. As noted, these numerical errors are minor, but forcing their cancellation in the budget is convenient for verifying conservation properties of the code. The evaluation of viscous transport using second differencing satisfies another conservation property, namely, the net change of TKE due to viscous transport of TKE exactly balances the viscous fluxes of TKE to the walls.

TKE production involves the mean velocity, which is nonzero only for the **u** component in statistically stationary channel flow. The non-vanishing contribution is then $-\Phi\frac{\partial\langle\mathbf{u}\rangle}{\partial z}$, where Φ denotes the flux of **u** fluctuations resulting from fluctuations of the wall-normal velocity. For generality, the evaluation of TKE production is coded to include contributions from velocity components **v** and **w** as well as **u** because they might have nonzero mean values in other applications.

The conventional representation of Φ as the Reynolds-stress component $\langle u'w'\rangle$ can be evaluated in ODT, but in ODT this quantity does not conform to the foregoing definition of Φ because eddy events rather than the **w** velocity component are the mechanism of advection in the wall-normal direction. Therefore the evaluation of Φ in ODT is based on the ODT advection mechanism, as follows.

As mentioned earlier, **width** times the sum of **eavg(j,1,1)** for all **j** exceeding some value j_0 is the eddy-induced flux of **u** across the boundary between cells j_0 and $j_0 + 1$, so it corresponds to the physical definition of Φ. Evaluation of Φ at cell centers requires interpolation, which is accomplished by summing for all **j** exceeding j_0 plus half of **eavg(j,1,1)** evaluated at j_0. Centered differencing is used to evaluate $\partial\langle\mathbf{u}\rangle/\partial z$ at cell centers. To evaluate the z derivative, the centered difference is divided by $2 \times$ **width**. Combined with the factor of **width** in the flux computation, the net effect is that **width** does not appear in the computation of the production term. The factor of 2 in the denominator accounts for the factor .5 in statements beginning with $\mathbf{p}(*) =$.

Evaluation of the production term at cell boundaries would be less useful because other terms in the TKE budget are evaluated at cell centers and the algorithm is set up to achieve balance to high accuracy at each node of the computational mesh. Another consideration is that the data reduction algorithm is designed to preserve statistical reflection symmetry of the simulated flow, enabling an additional verification of the code.

Owing to the stochastic rather than kinematic flow response to forcing mechanisms in ODT, there is no unambiguous way to discriminate the TKE budget terms conventionally denoted turbulent transport and pressure transport. It is possible only to identify their combined effect, here termed advective transport and operationally defined as the eddy-induced flux of TKE. In fact, the quantity most readily computed is the sum of the advective transport and production terms, from which the production term is subtracted to obtain the advective transport.

This sum of terms is simply the eddy-induced rate of change of TKE, because transport and production and the only mechanisms by which eddy events change TKE at a given location. What is needed, then, is half the average rate of change of v_i^2 minus $\langle v_i \rangle$ times the average rate of change of v_i, summed over components **i**. The identified quantities are provided in the data arrays **eavg(*,2,i)**, **cavg(*,i+1)**, and **eavg(*,1,i)**, respectively. Because time averages of **v** and **w** are zero for channel flow, not all terms are needed for all components for the present application, but all terms are included for generality.

6.3 Data output

Output is written in one of two formats, depending on the value of **ioptions(1)**, which has the default value 0 but can be assigned a different value that is read from the input file **BOptions.dat** in subroutine **BReadOptions**. In **BWriteSeries**, option value 0 gives a list of evaluation times from array **time**, followed by a list of variance values from array **temp**. Option value 1 gives a file format readable by XMGRACE, namely a list of records, each consisting of a time and the corresponding variance value. For either option, the output file, unit 40, is **T.dat**. For option value 0, the first record written to **T.dat**, ahead of the aforementioned data lists, is the number of entries in each list.

Likewise, in **BSnap**, option value 0 gives a list of values of the independent variables (here, wall-normal location) followed by one or more lists, each of which consists of corresponding values of a dependent variable. These lists are preceded by a record giving the number of entries in each list. This structure is implemented through the sequencing of calls of subroutine **BRecord**, which writes data to output files. If the argument **N** specifying the list length is negative, then the absolute value of **N** is written before the data list **h** is written (where **h** is based on data list **s**, which is an argument of **BRecord**). The purpose of data transfer of data from array **s** to array **h** is to process the data to prevent underflows when output is written.

In **BSnap**, option value 1 causes output to be written in XMGRACE-readable format in subroutine **XRecord**. Each written record consists of a wall-normal location and the corresponding value of a dependent variable.

Location zero is not included in the numerical representation of the flow state. For option value 1, each output file begins with a record written by a call of **XRecord** that consists of wall-normal location zero and the wall value of the dependent variable. For channel flow, all but two of the computed statistical quantities are zero at both walls. The exceptions are viscous transport of TKE, whose values at both walls are set equal to its computed value at location $\mathbf{N} - 1$, and viscous dissipation of TKE, which is evaluated at the walls in a way that forces it exactly to balance viscous transport of TKE.

BSnap is set up to write output for four averaging intervals (**nstat** = 4). All output file names have the suffix **.dat** and each name consists of a letter and a number denoting the averaging interval. For option 0, the file contents, keyed to the first character of the file name, are: A, mean streamwise velocity; B, velocity variances (for each of the three components); C, wall-normal advective flux of streamwise velocity; D, terms of the turbulent kinetic energy budget (shear production, advective transport, viscous transport, and dissipation). For option 1, A, B, and C have the same meanings, but the four budget terms are written to individual files starting with D, E, F, and G, respectively.

For either option, the sum of budget terms (with dissipation negative, so balance gives a null result) is written in option-1 format to a file with first character H. Also, eddy size histograms (see Sec. 6.2) are written to a file with first character I.

To monitor the progress of the simulation, **BInitRun** writes "Starting the run" and **Bodt** writes the new values of **iter** and **istat** whenever these indices are updated. The output buffer is flushed after each write statement so the writes, to output file **Brun.dat**, are done promptly.

7 Example

In the tar archives input*.tar and output*.tar, input and output files are provided for an illustrative case consisting of one realization (**niter** = 1) of channel flow for $\mathrm{Re}_\tau = 590$. This value of Re_τ was enforced by the choice of the pressure-gradient forcing term G = **pgrad**, defined below Eq. 1, for given values of channel half-height h = **dom**/2, and the kinematic viscosity ν = **visc**, based on the steady state momentum balance condition $\mathrm{Re}_\tau^2 = h^3 G/\nu^2$. ODT model parameters C and Z (see Eq. 2 in the Appendix) are chosen to provide reasonable agreement with the $\mathrm{Re}_\tau = 590$ channel flow direct numerical simulation results of Moser, Kim, and Mansour [4], but they have not been optimized based on a precise criterion.

Discussions of the dependence of ODT channel flow results on model parameters are provided in [1] and [5].

All velocity profiles are initially zero except for the small added random noise that is explained in the Appendix. Data is collected in four averaging intervals. The results for these intervals can be compared to see the effect of the initial transient in the first interval relative to the later intervals. In particular, the terms of the steady state TKE budget do not sum to zero if the TKE profile changes significantly during an averaging interval, as may occur during the initial transient relaxation period or because the averaging intervals are short enough so that the TKE rate of change, which is the difference between ending and beginning values divided by the elapsed time, is non-negligible because the denominator is not large enough.

The input files are described in Sec. 3, and additional information is provided in README and in comments in the subroutines that read the inputs. The output files and file formats are based on output option 1, as described in Sec. 6.3.

Because the illustrative case consists of one simulated realization (**niter** = 1), statistics gathered in **BSeries** and reduced in **BWriteSeries** are not meaningful. Due to the physical resolution requirement (order h/Re_τ) and the long physical time of the realization (chosen to produce high precision statistical results), the CPU time to run the illustrative case was roughly 50 hours. This can be reduced for quick turnaround by reducing Re_τ or the elapsed time per realization, **tend**.

The present code is specialized rather than multi-purpose, but it has more features and options than are needed to run the illustrative case. To date, the code has been tested only for the flow configuration and model formulation used for the illustrative case.

Acknowledgements

The code documented here is based on an ODT code originally developed by Scott Wunsch. Code development and documentation was supported by the U.S. Department of Energy, Office of Basic Energy Sciences, Division of Chemical Sciences, Geosciences, and Biosciences. Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under contract DE-AC04-94-AL85000.

Appendix: Theory and numerical implementation of eddy sampling and the eddy event

The physical motivation of the modeling approach is discussed in several references, so the emphasis here is on operational and numerical considerations. The eddy sampling procedure and the implementation of an eddy event are explained. The formulation used here is a specialization of the formulation described in [1] to constant-property flow.

Although eddy location and size are governed by integer variables **M** and **L** in the present numerical implementation, in theory (and in an adaptive-mesh implementation not described here) eddy location and size are real numbers, here denoted z_0 and l. In this framework, an eddy rate distribution $\lambda(z_0, l; t)$ is defined, where λ has a different meaning than in Sec. 5.2. Here, $\lambda(z_0, l; t)\, dz_0\, dl$ is the rate of occurrence of eddies whose left boundary is in the range $[z_0, z_0 + dz_0]$ and whose size is in the range $[l, l + dl]$. This rate depends on the evolving flow state, hence varies with time, as indicated by the t dependence of λ. For any set of eddies with a rate of occurrence defined in this manner, eddy occurrences are assumed to be statistically independent events (Poisson event sequence) occurring with the specified rate (which is time varying, as noted).

Section 5.2 defines a property of the triplet map called the kernel, here denoted $K(z)$. For given z_0 and l, $K(z) = z - \hat{z}$, where $\hat{z}$ is the location mapped to z by the corresponding triplet map. Note that $K(z) = 0$ if z is not in $[z_0, z_0+l]$. Like the eddy location and size, the triplet map has a continuum definition whose specialization to a discrete uniform mesh is used in Sec. 5.4. The continuum definition of the map is not stated here though its use is implied in what follows.

A key quantity used to specify λ is $s_K \equiv (1/l^2) \int s(\hat{z})K(z)\, dz$, where s is a placeholder for any evolving flow property (in the present formulation, a velocity component). There are several efficient ways to compute s_K numerically. In function **BsKd** of the present code, the triplet map is applied to the property profile s to obtain a mapped profile s', where by definition, $s'(z) = s(\hat{z})$. Then $s_K = (1/l^2) \int s'(z)K(z)\, dz$ is used to evaluate s_K.

In **BsKd**, the variable **f** corresponds to the property s for which s_K is evaluated. **f** is copied to the scratch array **s** over the argument range in which K is nonzero (except that K is zero at the each end of this range in the current implementation). Within this range, the discrete triplet map is applied to **s** in subroutine **BTriplet**.

For numerical implementation of the definition of s_K, the integral over dz is expressed as a sum of mesh cells times the cell size **dom/N**, and l and K are each expressed as an integer number of mesh spacings times the cell size, e.g., $l = \mathbf{L} \times \mathbf{dom/N}$. Factors of **dom/N** in the numerator and denominator cancel, so the numerical result has the same units as s (m/s in the present implementation).

In **BsKd**, the summation is evaluated in a loop over the number **Lo** of cells in each of the three parts ('images') of the triplet map implementation described in Sec. 5.4. Each pass through the loop increments the sum based on the contributions of three cells, one from each image. Displacements **y1**, **y2**, and **y3** numerically implement K based on the discrete triplet map as described in Sec. 5.4. **j1**, **j2**, and **j3** are the indices of cells contributing to the sum during the current pass through the loop. After the summation is completed, the sum is divided by $\mathbf{L} \times \mathbf{L}$ to complete the evaluation of **BsKd**.

A key modeling assumption is the choice of a mathematical expression for λ, namely

$$\lambda = (C/l^3)[u_K^2 + v_K^2 + w_K^2 - Z(\nu/l)^2]^{1/2}, \tag{2}$$

where C and Z are free parameters of the model. In the numerical implementation, the discrete counterparts **M** and **L** of z_0 and l respectively can be interpreted as values associated with discrete bins in the two-dimensional continuum of z_0 and l values. The width of an **M** bin is the mesh spacing, **dom/N**, but the width of an **L** bin is three times this value because **L** must be an integer divisible by three. On this basis, the rate $\Lambda(\mathbf{M}, \mathbf{L})$ of occurrence of eddies for given **M** and **L** is taken to be $3(\mathbf{dom}/\mathbf{N})^2\lambda(z_0, l)$, where z_0 has the value $\mathbf{M} \times \mathbf{dom}/\mathbf{N}$, l has the value $\mathbf{L} \times \mathbf{dom}/\mathbf{N}$, and the dependence on t is suppressed. Here, the factor multiplying λ is the discrete analog of the differential $dz_0\, dl$.

A more careful treatment would involve integration of λ over the range of its arguments within a given bin. This is impractical and it is not needed because either treatment converges to the continuum mathematical definition of the model as the mesh is refined, and the simpler treatment gives sufficently rapid convergence in sensitivity studies.

Equation 2 is substituted into the relation between Λ and λ, and all occurrences of l in Eq. 2 are replaced by $\mathbf{L} \times \mathbf{dom}/\mathbf{N}$, giving

$$\Lambda(\mathbf{M}, \mathbf{L}) = [(3C\mathbf{N}/\mathbf{dom})/\mathbf{L}^3][u_K^2 + v_K^2 + w_K^2 - Z(\nu\mathbf{N}/\mathbf{dom})^2/\mathbf{L}^2]^{1/2}. \tag{3}$$

In subroutine **BReadConfig**, $3C\mathbf{N}/\mathbf{dom}$ is evaluated and set equal to the variable **ratefac**, and $Z(\nu\mathbf{N}/\mathbf{dom})^2$ is evaluated and set equal to the variable **viscpen**. (In **BReadConfig**, C, Z, and the kinematic viscosity ν are represented by variables **C**, **Z**, and **visc**, respectively.) The values of **ratefac** and **viscpen** are returned to subroutine **Bodt** and are then passed down the calling chain to subroutine **BProb**.

The subroutine **BSampleEddy** passes $\mathbf{L3} = \mathbf{L}/3$, rather than **L**, to **BProb**, so **BProb** begins with the computation of **L** from **L3**. Then **uK**, **vK**, and **wk**, corresponding to u_K, etc., are computed using function **BsKd**. Using these, **viscpen**, and **L**, the argument of the square root in Eq. 3 is evaluated and set equal to the variable **p**.

Negative **p** implies an imaginary value of the rate Λ of eddy occurrence for the specified **M** and **L** values. Physically, the term containing the factor Z represents an eddy suppression mechanism, and negative **p** corresponds to dominance of this mechanism over eddy generation, which is governed by the other terms. (The definition of s_K indicates that the eddy generation terms, which are non-negative, are associated with spatial variation of velocity components; the specific definition has useful physical and mathematical properties for modeling purposes.) For this reason, the variable **pp**, which is the acceptance probability (subject to possible modification in **BSampleEddy**; see Sec. 5.2), is set equal to zero if **p** is not positive. Otherwise, the evaluation of **pp** continues.

As explained in Sec. 5.2, **pp** is the desired sampling rate λ of a particular type of eddy divided by its actual sampling rate, where the latter is expressed as the total sampling rate λ^* of all eddies times the probability h of selecting particular **M** and **L** values. For the Poisson sampling process assumed here, λ^* is the inverse of the sampling increment **dt**. Section 5.2 does not distinguish between the continuum theoretical and the spatially discrete numerical eddy rate distribution, but here this distinction is made, so $\mathbf{pp} = \Lambda \mathbf{dt}/h$. Accordingly, the evaluation of **pp** involves the factors $\Lambda = (\mathbf{ratefac}/\mathbf{L}^3)\sqrt{\mathbf{p}}$, **dt**, factors that evaluate h, and an additional factor $\mathbf{disfac} = 1 - 3/\mathbf{L}$.

disfac is a numerical correction to the theoretical eddy rate distribution, Eq. 2. It accounts for the fact that the **L** value of an eddy of given physical size depends on spatial resolution, e.g., the choice of **N** for given domain size **dom**. The simulation is intended to approximate the limit of infinite resolution, so deviations from this limit are numerical errors. One numerical error arises because the mean square fluid displacement by an eddy consisting of **L** cells that has a given physical size has the **L** dependence given by **disfac**. (This can be shown by mathematical induction for even **L** by starting from $\mathbf{L} = 6$ and for odd **L** by starting from $\mathbf{L} = 9$.) In other words, the mean square displacement is smaller by the factor **disfac** than in the limit of infinite resolution, with commensurate reduction of the turbulent transport induced by eddies consisting of **L** cells. To compensate for this numerically caused transport reduction, **pp** is divided by **disfac**. The resulting increase in the eddy sampling rate counteracts the numerical bias so that the turbulent transport is the same as in the continuum limit.

Strictly speaking, this correction is not needed because resolution should in all instances be chosen to obtain a mesh-independent solution. However, correction for mesh biases accelerates convergence to this solution as resolution is improved, so a converged solution is obtained using a coarser mesh than would otherwise be needed, implying a less costly computation.

As explained in Sec. 5.2, h is taken to be the product of probability distributions of **M** and **L**, where **M** is sampled uniformly from $[1, \mathbf{N}-\mathbf{L}]$. Therefore the probability of any **M** value is $1/(\mathbf{N} - \mathbf{L})$, hence the factor $\mathbf{N} - \mathbf{L}$ appearing in the numerator of the statement that evaluates **pp**. In Sec. 5.2 it is also explained that **PL** is the cumulative distribution of **L**, and that the difference between successive values of the array **PL** is the probability of a given **L** value, which is one of the terms in the evaluation of **pp**.

The array **PL** is evaluated once before the simulation begins in subroutine **BLenProb**. Its evaluation is based on the assumed functional form

$$f_c(s) = A_c s^{-2} \exp(-2s_p/s) \tag{4}$$

of the probability density function of s, which is a continuous (hence subscript c) variable corresponding to one-third of the eddy length l. s_p is the value of s for which $f(s)$ is largest (i.e., the most probable s value). Equation 4 applies only within a specified s range, $[s_o, s_v + 1]$, where s_o and s_v are integers. Outside that range, $f(s) = 0$. The normalization factor A_c is determined by the requirement that the integral of $f(s)$ must be unity, giving

$$A_c = 2s_p/[\exp(-2s_p/(s_v+1)) - \exp(-2s_p/s_o)]. \tag{5}$$

In the numerical implementation, the probability $q(S)$ that s has the integer value S is taken to be the integral of $f(s)$ from S to $S+1$, giving

$$q(S) = \frac{A_c}{2s_p}\exp(-2s_p/S)[\exp[2S_p/(S(S+1))] - 1] \tag{6}$$

for $s_o \le S \le s_v$. $q(S)$ is zero for $S < s_o$ and $S > s_v$.

In **BLenProb**, $A_c/(2s_p)$ is replaced by a coefficient **C** (unrelated to the eddy rate parameter) that is set equal to the inverse of the sum over S of the quantity **z** that multiplies **C**. In **BLenProb**, the summation index is **I** rather than S and the bounds are **Io** and **Iv** rather than s_o and s_v. Likewise, **Ip** corresponds to s_p. The parameters **Io**, **Iv** and **Ip** are assigned in **BLenProb**, but these assignments are superseded by whichever of these quantities are read from input during code initialization, as controlled by the variable **nipars** (itself read in **BReadPars** during initialization). Input values, if any, are in elements 3 through 5 of array **ipars**.

On this basis, the partial sums of $q(S)$ are evaluated in **BLenProb**, forming the array **PL** that is used to sample S values. The assumptions, e.g., Eq. 4, on which this method are based are convenient for numerical implementation and allow efficient sampling when good choices of **Io**, **Iv** and **Ip** are found based on prior experience, trial and error, or statistics gathered in test runs. **Iv** can be used as a physical model parameter to test the effect of omitting the largest eddies that the domain can accommodate.

One convenient feature of the sampling method is that there is a simple continuum approximation of the cumulative distribution **PL** that can be inverted in closed form to estimate the sampled S value, namely,

$$\mathbf{x} = -2s_p/\ln\left[\frac{2rs_p}{c} + \exp(-2s_p/s_o)\right], \tag{7}$$

where **x** is the estimate of s and r is the corresponding cumulative probability. In **BLenProb**, variables **Co** and **Cv** are evaluated that facilitate the computation of **x** in **BLength**. In that computation, random sampling from the cumulative distribution is performed by assigning r to be the variable **rannum** that is sampled, in subroutine **Brng**, from the uniform distribution over $[0,1]$. As explained in Sec. 5.2, **x** is used to obtain an initial guess from which the sampled value of **I** is determined by an iterative procedure.

Turning to eddy implementation, details of the use of subroutine **BAddK** in subroutine **BEddy** are explained. Variables **y1**, **y2**, and **y3** are kernel values expressed as in function **BsKd**, and the corresponding array indices **j1**, **j2**, and **j3** are likewise expressed as in **BsKd**. **BAddK** adds **c** times this array of kernel values to an input array. In the calling routine, **c** has different values, **cu**, **cv**, and **cw**, in applications of this operation to arrays **u**, **v**, and **w**, respectively.

The purpose of this operation is to distribute, equally among these velocity components, a property termed 'available energy' that is defined within the modeling framework. Addition of $\mathbf{c}K$ to a generic velocity profile s changes the total kinetic energy of that velocity component, where the new kinetic energy is proportional to the integral, $I(\mathbf{c})$, of $(s + \mathbf{c}K)^2$ over $[z_0, z_0 + l]$. I has a quadratic dependence on $\mathbf{c}$ and has a minimum value $I_{\min}$ for some finite $\mathbf{c}$ value. $I(0) - I_{\min}$ is the largest possible reduction of I that can be achieved by adding $\mathbf{c}K$ to s. The difference $\Delta = I(0) - I_{\min}$, multiplied by half the fluid density, is the available energy of velocity component s. (Dimensionally, it is energy per unit surface area normal to the z direction.)

The eddy event consists of two operations, a triplet map followed by available energy redistribution among velocity components. For this purpose, Δ is evaluated after the triplet map is applied, giving $\Delta_s = s_K^2 / \int K^2\,dz$, where the subscript s is a velocity component label.

The empirical principle that turbulence promotes a tendency toward local isotropy suggests that the available energies of the three velocity components should be equalized. This is accomplished by choosing values **cu**, **cv**, and **cw** that result in equalization of Δ_u, Δ_v, and Δ_w. Conservation of energy requires the sum of these quantities to be the same before and after equaliztion, so after equalization, each of them must attain the value $\mathbf{root}^2 / \int K^2\,dz$, where the variable **root** is evaluated as $\left[\left(\mathbf{uK}^2 + \mathbf{vK}^2 + \mathbf{wK}^2\right)/3\right]^{1/2}$ in **BEddy**. For a given component s, these assumptions and relations yield $\mathbf{c} = (l^2 / \int K^2\,dz)(-s_K \pm \mathbf{root})$. To choose between the two solutions for $\mathbf{c}$, it is reasonable to require $\mathbf{c}$ to be zero if all three velocity components are identical, because then they all have the same available energy so no modifications are needed. In this case, $\mathbf{root} = |s_K|$. To force $\mathbf{c}$ to vanish when this holds, the solution branch is specified by expressing $\mathbf{c}$ as $(l^2 / \int K^2\,dz)[-s_K + \mathrm{sgn}(s_K)\mathbf{root}]$. This is the expression used to evaluate **cu**, **cv**, and **cw** in **BEddy** with the appropriate substitutions for s_K.

As in the evaluation of s_K, both K and l in this expression are evaluated as integer numbers of mesh spacings times the cell size, and the integral is expressed as a sum over mesh values times the cell size, yielding a net -1 power of cell size. The latter is cancelled by a factor of cell size coming from K when $\mathbf{c}$ is multiplied by K in subroutine **BAddK**. Therefore it is consistent to evaluate K and l as integer numbers of mesh spacings (hence replacing l by $\mathbf{L}$) in both **BEddy** and **BAddK** without introducing any factors of cell size. On this basis, the definition of the discrete triplet map yields the result $(4/27)\mathbf{L}^3(1 - 3/\mathbf{L})$ for the discrete sum corresponding to $\int K^2\,dz$. Based on these results, $l^2 / \int K^2\,dz$ is evaluated as $(27/4)/[\mathbf{L}(1 - 3/\mathbf{L})]$, corresponding to the variable **cfac** in **BEddy**.

In the evaluation of $\mathbf{c}$, $\mathrm{sgn}(s_K)$ is ambiguous if $s_K = 0$ for a particular component s, which matters if **root** is nonzero. An arbitrary choice is inconsequential if this scenario is rare. To assure that this scenario rarely occurs, velocity profiles that are nominally uniform initially (a condition that gives vanishing s_K) are perturbed initially by a small-amplitude random noise. This is implemented in the velocity profile initialization in subroutine **BInitRun**, where the initial noise is hard-wired to be 10^{-8} times an array of independent random samples from the uniform distribution over $[0, 1]$. The noise is applied to all three velocity components because they are all nominally uniform (in particular, zero) initially in the present set-up.

References

[1] Kerstein, A. R., "ODT: Stochastic Simulation of Multi-Scale Dynamics," in Interdisciplinary Aspects of Turbulence, W. Hillebrandt and F. Kupka, Eds., Lecture Notes in Physics (Springer, London, in press).

[2] Kerstein, A. R., Ashurst, W. T., Wunsch, S., and Nilsen, V., "One-dimensional turbulence: vector formulation and application to free shear flows," J. Fluid Mech. **447**, 85-109 (2001).

[3] Law, A. M., and Kelton, W. D., Simulation Modeling and Analysis (McGraw-Hill, New York, 2000).

[4] Moser, R. D., Kim, J., and Mansour, N. N., "Direct numerical simulation of turbulent channel flow up to Re_tau = 590," Phys. Fluids **11**, 943-945 (1999). Data downloadable from http://turbulence.ices.utexas.edu/MKM_1999.html

[5] Schmidt, R. C., Kerstein, A. R., Wunsch, S., and Nilsen, V., "Near-wall LES closure based on one-dimensional turbulence modeling," J. Comput. Phys. **186**, 317-355 (2003).