One-dimensional turbulence (ODT): computationally efficient modeling and simulation of turbulent flows

Victoria B. Stephens, David O. Lignell*

Chemical Engineering Department, Brigham Young University, Provo, UT 84602, USA

Abstract

One-dimensional turbulence (ODT) is an accurate and computationally efficient ~~technique~~ model for ~~modeling~~ simulating turbulent flows. ODT has been applied to a wide range of flow problems including reaction, multiphase, differential diffusion, heat release, buoyancy, and wall flows. Applications include use as a standalone model and as a closure for large-eddy simulation (LES). Its strength lies in the ability to capture a full range of turbulent length and time scales. The ODT model is strongly coupled with its implementation, complicating its formulations. We present a modern, open-source, object-oriented C++ implementation of ODT. ~~This code can be used as a starting point to understand, apply, and extend the ODT model, enabling its further application to turbulent flow research.~~ The code described here and made available online can be used as a starting point to understand, apply, and extend the ODT model, enabling its further application to turbulent flow research.

Keywords: turbulence, reacting flows, one-dimensional turbulence

*Corresponding author.
Email address: davidlignell@byu.edu (David O. Lignell)

Code Metadata

Nr.	Code metadata description	Please fill in this column
C1	Current code version	1.0
C2	Permanent link to code/repository used for this code version	https://github.com/BYUignite/ODT
C3	Code Ocean compute capsule	N/A
C4	Legal Code License	MIT
C5	Code versioning system used	Git
C6	Software code languages, tools, and services used	C++, Python 3.x, YAML,
C7	Compilation requirements, operating environments & dependencies	CMake 3.12+, Cantera, Git, Doxygen (optional)
C8	If available Link to developer documentation/manual	ODT style guide
C9	Support email for questions	davidlignell@byu.edu

Table 1: Code metadata (mandatory)

1. Motivation and significance

Turbulent flows characterize the vast majority of fluid flows in practical engineering applications, and simulations of turbulent flows provide researchers with valuable insights into complex systems, particularly reacting turbulent flows such as combustion processes. Turbulence is a complex phenomenon that affects the full range of a flow's length and time scales. As a result, resolving the entire flow field by numerically solving the Navier-Stokes equations of fluid flow, as is done in direct numerical simulations (DNS), requires substantial computational resources. DNS is a powerful research tool, but its high computational cost makes it intractable for simulating most practical engineering flows. In order to achieve numerical solutions to practical flow problems, researchers use alternative frameworks that model turbulence rather than resolving it directly.

Large-eddy simulations (LES) address the problem of wide-ranging length and time scales by combining direct resolution of grid-scale quantities, as in DNS, with subgrid modeling of smaller turbulence structures. The more complex the flow, the more modeling is required; for example, a jet flame simulation might require subgrid modeling for the combustion chemistry, radiative heat transfer, and soot chemistry in addition to turbulence structures, all of which form a tightly coupled system in which each model interacts heavily with the others. While subgrid modeling makes LES more computationally

affordable than DNS, it can introduce empiricism into simulations, which can lead to inaccurate results. Additionally, unresolved quantities are often parameterized in state space with empirical relationships or assumed distributions that lack universal applicability. LES is a valuable simulation tool, but its approach to turbulence modeling can introduce unwanted empiricism and make errors difficult to isolate and quantify.

The one-dimensional turbulence model (ODT) functionally reverses the LES approach, modeling large-scale turbulent advection and directly resolving small-scale flow structures, simulating the full range of length and time scales in a single dimension. Depending on the configuration, large-scale structures can be easier to model than small-scale structures, especially in reacting flows. ODT's direct resolution of fine scales in one dimension bypasses subgrid modeling issues that complicate LES. Previous studies show that ODT can attain accuracy comparable to DNS at a fraction of the computational cost [1, 2], making it an attractive tool for simulating turbulent flows. Because the ODT model is one-dimensional, it is limited to homogeneous or boundary-layer flows, such as jets, wakes, and mixing layers, but these types of flows are common in nature and central to turbulence research. ODT's computational efficiency and resolution of a full range of scales make it a valuable tool that complements experimental studies and other simulation tools like DNS and LES.

Early applications of ODT focused on homogeneous turbulence, wakes, and mixing layers [3, 4, 5]. Later extension to variable-density flows and a spatial downstream coordinate system facilitated its growth and application to more complex flows, including combustion in jet flames [6, 7, 8, 9, 10, 11, 12, 13], counterflow flames [14], wall fires [15], and sooting flames [1, 16, 17, 18, 19], as well as other particle flows [20, 21, 22, 23]. ODT has also served to complement LES through subgrid modeling studies [24, 25, 26] and has been applied to various other flow configurations such as double-diffusive interfaces [27], Rayleigh-Taylor mixing [28], and stratified turbulence [29]. Most recently, ~~the ODT code was~~ ODT implementations have been extended to include cylindrical and spherical coordinate systems [30, 31, 32].

~~During the recent implementation of the cylindrical and spherical model formulations, the ODT code was drastically overhauled and reorganized, resulting in its current configuration.~~ Many of the references cited above make use of a particular software implementation of the ODT model, referred to here as the *ODT code*. During the recent implementation of the cylindrical and spherical model formulations, this code was drastically overhauled and reorganized. The ODT code presented here is a pared down version of the development code, representing the fundamental aspects of the ODT model and its most reliable functions. The example case in Section 3 is a

representative sample of the ODT code's capabilities as it is presented here. Future releases will expand this code's functionality with additional features currently in development.

2. Software description

2.1. Model description

The ODT model formulation is described in detail in the literature [3, 5, 33, 30, 34]; only a brief explanation will be given here. In ODT, turbulent advection is modeled with stochastic processes called eddy events, which punctuate the solution of unsteady, one-dimensional transport equations for mass, momentum, and enthalpy. The ODT code implementation of the ODT model uses a Lagrangian finite-volume formulation for diffusive advancement in which mass stays constant within each grid cell while cell volumes increase or decrease according to cell dilation via an adaptive mesh refinement [34].

Transport equations for mass, momentum, and enthalpy in the temporal formulation of ODT take the following generic form, derived from the Reynolds Transport Theorem [35] for a given scalar quantity per unit mass β:

$$\frac{\mathrm{d}\beta}{\mathrm{d}t} = -\frac{j_{\beta,e}A_{x,e} - j_{\beta,w}A_{x,w}}{\rho V} + \frac{S_\beta}{\rho V}. \tag{1}$$

The first and second terms on the right-hand-side of this equation are the diffusive transport and source terms, respectively. Here, the term *diffusive* is not specific to mass transfer, but may refer to the transport of, e.g., heat, mass, and momentum, depending on the identity of the scalar β. In practice, we refer to the transport term as the "mixing term'. The symbols shown in Equation 1 are defined as follows: j_β is the diffusion flux of scalar β across the cell face area A_x, where the subscripts e and w refer to the "east" and "west" faces of a given grid cell, respectively; S_β is the Lagrangian source term derived from the conservation law for β; ρ represents mass density; and V represents cell volume. The generic transport equation differs slightly in the spatial formulation of ODT, but its form is similar, so we omit it here for brevity. The system of ordinary differential equations (ODEs) that results is well behaved at all grid points and in all geometries in their finite-volume forms. For details on transport equation derivation and use in both the temporal and spatial formulations of ODT, see [30].

Eddy events occur as a Poisson process in accordance with their eddy rates, where a given eddy event of size l and location x_0 has an eddy timescale t and an associated eddy rate $1/t$. Three user-defined ODT parameters control the eddy event process: the eddy rate parameter C scales the rate of occurrence of the eddies; the viscous penalty parameter Z suppresses small

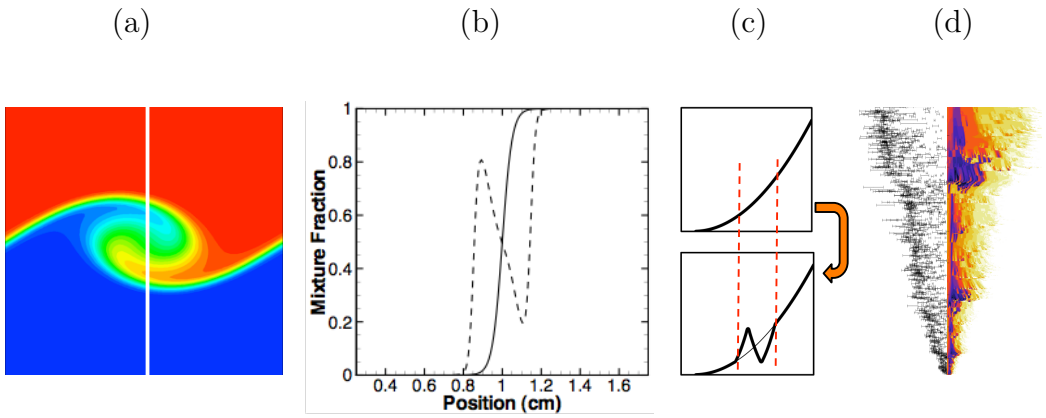

Figure 1: Schematic diagram of a triplet map. From left to right: (a) simulation of a single "eddy", a two-dimensional Kelvin-Helmholtz instability with a vertical white line-of-sight; (b) mixture fraction profile along the indicated line-of-sight before and after the eddy showing increased gradients and surface area; (c) a triplet map operates on a profile to produce a mapped profile with three copies, compressed and lined up, with the middle copy spatially inverted; (d) illustrative eddy event sizes and locations with corresponding temperature profile for a single ODT realization of a wall fire, adapted from [15].

eddies; and the large eddy suppression parameter β_{LES} constrains eddies to those with timescales proportional to the elapsed simulation time. Sampled eddies that do not fit the defined parameters are rejected and not applied to the domain.

Eddy events modify domain variables using "triplet maps", as illustrated in Figure 1. In an eddy region of size l, transported property profiles are modified (mapped) by making three copies, compressing each spatially by a factor of three, and realigning them with the middle copy spatially reversed. This map is continuous, conserves all quantities and statistical moments, increases scalar gradients, and decreases length scales consistent with turbulent eddies in real flows. Subsequent eddies in the same region will result in a cascade of scales. The eddy rates, used to sample and select eddy events, depend on eddy size and the local kinetic energy consistent with turbulent scaling laws.

Eddy events occur concurrently with diffusive advancement via solution of the system of unsteady one-dimensional transport equations. In this way, the ODT code marches in time or space until it reaches the simulation end point. Due to the stochastic nature of eddy events, each ODT simulation, or realization, is different, even when it is provided with the same input parameters. In order to obtain statistically stable data for a given set of parameters, we run many realizations with the same input parameters and

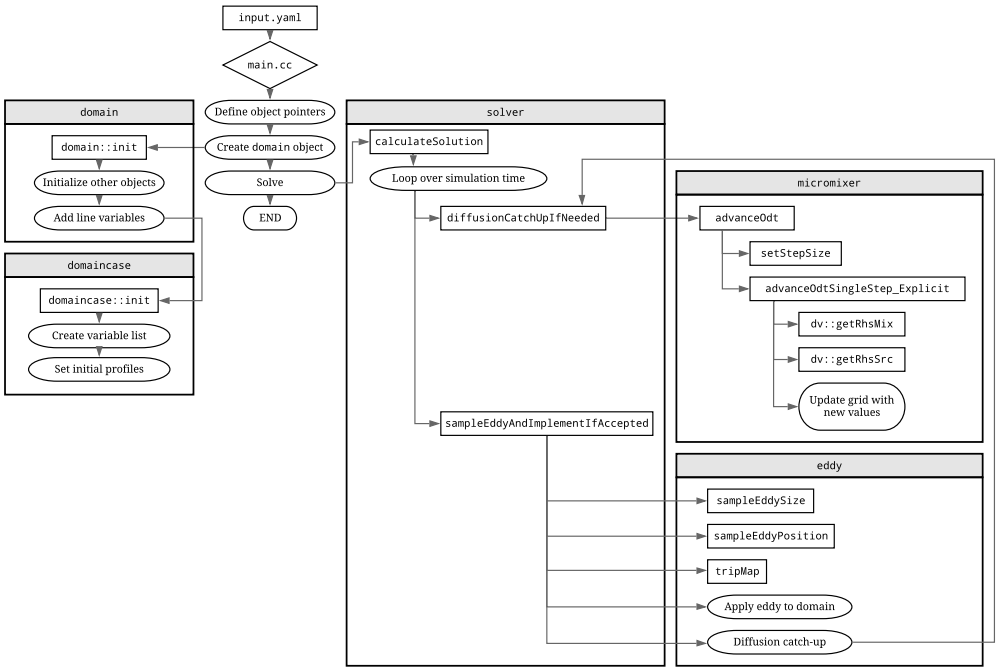

Figure 2: Structural outline of the ODT code, including its major class objects. For brevity, this diagram makes two simplifications. First, it assumes that diffusive advancement in the `micromixer` uses an explicit solution method; in practice, users specify either an explicit, semi-implicit, or Strang split method. Second, it implies that all of the eddy events sampled by the `eddy` object are applied to the domain; in reality, eddy events are filtered through several rejection tests (omitted from this figure for clarity) before acceptance and application to the domain.

then gather statistics over these realizations. This is done via post-processing tools. Post-processing is specific to a given case and the needs of the user. Examples of post-processing files are included for different cases provided with the ODT code, as discussed further in Section 2.3.4.

2.2. Software Architecture

The ODT code package consists primarily of an object-oriented C++ code responsible for running flow simulation cases and generating data. The package also contains auxiliary data processing and visualization tools, written mostly in Python. The post-processing tools are case-specific but will be addressed generally in Section 2.3.

Figure 2 illustrates the ODT code's most important objects and structural features. User inputs are provided to the executable in YAML [36] format via

`input.yaml`; the location of the specific `input.yaml` file to be used is determined by the case name and case type specified in the run script. The `main` function defines storage for the main objects, but, once created, the `domain` object is responsible for object initialization as well as variable initialization and storage via a case-specific `domaincase` object.

Three primary objects handle the code's main functions: the `solver`, `micromixer`, and `eddy` objects. The `solver` coordinates the ODT solution process, marching along the simulation time and invoking diffusive advancement and eddy events. The `micromixer` handles diffusive advancement by setting step sizes, interacting with the transported domain variables, and solving the system of ODEs defined by Equation 1 (or its equivalent in the spatial formulation). The `micromixer` includes three solution methods that can be specified in `input.yaml`: a first-order explicit Euler method (pictured in Figure 2), a first-order semi-implicit method, and a second order Strang splitting method [37]. The latter two methods are useful for treating stiff chemistry, with source terms integrated using CVODE [38]. In reacting flow cases, enthalpy and chemical species mass fractions are transported, and physical properties and kinetic rates are computed with Cantera [39]. The `micromixer` is also the code's primary point of interaction with the `mesher` object (not pictured in Figure 2), which manages the adaptive grid functions. Finally, the `eddy` object manages eddy event selection and implementation, which proceeds as described in Section 2.1.

2.3. Workflow

This section outlines the process a user goes through in order to successfully build the ODT code and run a simulation. The ODT code is a standalone, self-contained package, and users interact with its files primarily via the command line rather than a graphical interface. For more details, please see the package documentation.

Within the main download package, several directories organize the ODT code. The `source`, `build`, and `run` directories contain the ODT source code, compilation tools, and run scripts, respectively. The `input` directory contains subdirectories corresponding to several possible case types, each populated with the appropriate input files. The `data` directory, initially empty, holds the raw data files and runtime information generated ~~by ODT~~, as well as post-processed data files generated from within the `post` directory. Finally, the `docs` directory contains documentation optionally generated by Doxygen [40] during the build process.

2.3.1. Building ~~ODT~~ the ODT code

The ODT code build process is automated with CMake. The user navigates to the `build` directory and edits the CMake configuration file. The CMake configuration file specifies Cantera's location and must be changed to reflect the local installation location. For reacting flows, the chemical reaction rates can be computed by Cantera (default), or a from a user-specified function, specified at compile time. Once the configuration settings are updated, the user runs CMake to apply the changes.

The `yaml-cpp` package is required in order to process input files. If it has not yet been installed, it must be built and installed at this point. For convenience, the ODT code package automates this process. YAML need only be installed once for a given instance of the ODT package. Rebuilding the ODT code with CMake does not affect the YAML installation.

Once CMake and YAML have been prepared, the user can build the ODT code with the `make` command, which places the code executable in the `run` directory. The most common errors that occur during the build process involve incorrect file paths or incomplete installation of required packages. See the build documentation for troubleshooting help.

Once the code is built, the user may optionally build a local copy of the documentation via Doxygen, which must be previously installed. This step is not required in order to run the ODT code. As an alternative, documentation can be accessed at the code repository wiki.

2.3.2. Input files

User-modified input files are located within the `input` directory, which contains subdirectories that correspond to various case types that can be run ~~with ODT~~. At minimum, a case's subdirectory must contain an `input.yaml` file, but may contain other files, or subfolders with supplementary information.

The `input` directory also contains the `gas_mechanisms` subdirectory, which contains chemical mechanism files that can be used in reacting flow cases. For cases in which a chemical mechanism is not required, the `not_used.xml` mechanism file is specified in `input.yaml`. Note that the chemical mechanism chosen for the case and specified in the input file must match the mechanism specified in the CMake configuration file during the ~~ODT~~ build process.

Input files contain simulation parameters in a human-readable format parsed by YAML. Within `input.yaml`, parameters are organized into sections, several of which are common to all input files. Parameter values present in the provided input files may be considered defaults that can be used to run a successful simulation of that case type. General code parameters are

stored in the `params` class. Variables needed in a simulation but not provided in an input file (not recommended), are either given defaults specified in the `params` class or result in an error message. Details about individual parameters, including usage and typical values, are covered in the documentation.

2.3.3. Running ~~ODT~~ the ODT code

To run a simulation, users must then navigate to the `run` directory, which contains the `odt.x` executable and several possible run scripts. The simplest option is `runOneRlz.sh`, which runs one realization of ODT in the specified configuration. In this run script, the user must alter two variables near the top of the file: `inputDir`, which specifies which input directory and files to use; and `caseName`, which provides a name for the simulation and the data files it outputs. The `runManyRlz.sh` script runs many realizations in serial, one after the other; it differs from `runOneRlz.sh` only in that the user must also alter the `nRlz` variable, which specifies the number of realizations to run. Each realization has a different seed value, which can be randomized or directly specified in the input file. To run the simulation with either `runOneRlz.sh` or `runManyRlz.sh`, save the run script and execute it at the command line. Users will see some output on the command line, but no data, which is instead output to the `data` directory.

ODT simulations can also be run in parallel using MPI. Two run scripts, `slrmJob.sh` and `slrmJob_array.sh`, are configured using Slurm [41], a common workload manager used for massively parallel computing resources. This allows many ODT realizations to run in parallel rather than in serial, reducing overall simulation time. Individual realizations are independent (embarrassingly parallel), but users must take care with case names and input file changes to ensure that individual realizations or entire cases are not overridden accidentally.

2.3.4. Data files and post-processing

For a given simulation, data is output to the `data` directory, which contains subdirectories for each simulation, specified by the `caseName` variable in the run script. Figure 3 illustrates the structure of the `data` directory and the locations of files within it. Each case folder is subdivided into `input`, `runtime`, `data`, and `post`. The `input` folder contains a copy of the input files used for the simulation; `runtime` contains runtime output, including eddy event information; `data` contains the raw data files; and `post` contains post-processed data files once they are generated.

To use post-processing tools, users navigate to the `post` directory. Within the `post` directory, data processing tools are organized by case type, which

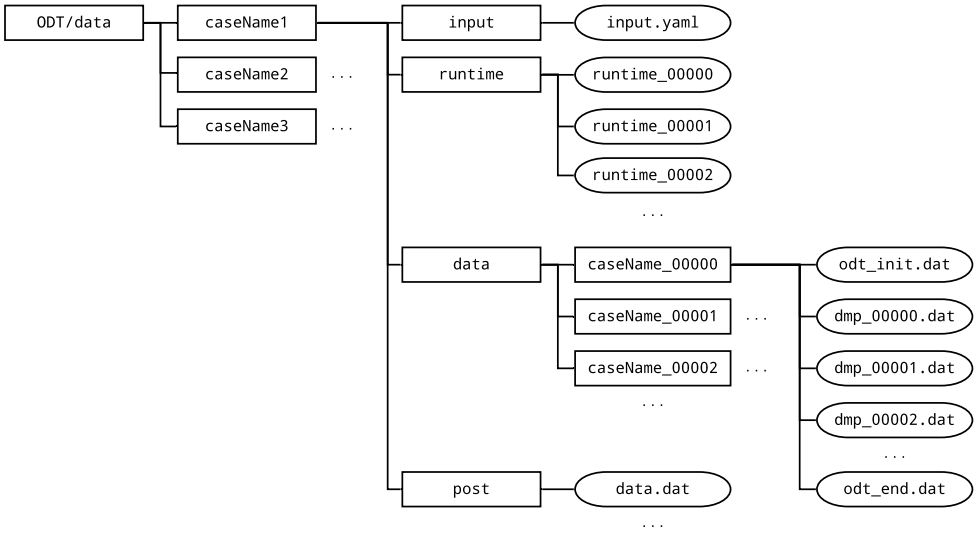

Figure 3: Data folder structure and file locations. Boxes indicate folders and ovals indicate files. For a given simulation, `caseName` will be replaced with the case name specified in the run script. The number attached to `caseName` folder names and to `runtime` data files indicates the realization number. The number attached to `dmp` files indicates the index of the list of "dump" times included in, or generated from the information in `input.yaml`.

is specified in `input.yaml` and determines case-specific variables and simulation setup parameters (refer to the `domaincase` object in Figure 2). Each set of post-processing tools is different, but may contain some combination of Jupyter notebooks, Python scripts (often coordinated by a `driver.py` file), experimental data files for comparison and plot generation, or other supplementary files. Post-processed data and generated plots are deposited in the `data` directory, within the appropriate `caseName/post` folder.

As part of the post-processing tools, there is a `post/tools` directory that contains two files: `data_py.py` and `data_tools.py`. The first of these can be used to convert the text data files generated during a run into binary numpy files. This can reduce both the number of data files and the file sizes. The number of files is reduced since all code realizations at a given output time are saved in the same binary file. The `data_tools.py` file is a module that contains a number of functions for processing the data files, such as functions to

- get the data from a given realization,
- return the number of realizations,
- get a list of the variable names in the data files,
- compute axial locations in the spatial ODT formulation,
- query a parameter from the input file,
- return the stoichiometric mixture fraction for combustion/mixing problems, and
- compute the domain bounds.

In addition, a few functions are provided for doing interpolation/extrapolation, and computing probability denisity functions (PDFs), and weighted PDFs. Many of these functions take an argument that is a dictionary of case-specific information, such as the case name and paths for key directories. This dictionary is set in the post-processing driver script based only on the user-specified case name.

3. Example Case: Canonical Jet Flame

~~ODT~~ The ODT model is uniquely suited to reacting flow simulations. Here, we present illustrative ODT simulation results of a round, turbulent jet flame based on and compared to the experimental DLR-A flame of Meier

et al. [42]. This canonical flame configuration has been used extensively to study and validate turbulent combustion models [43, 44, 45, 46, 47, 48].

The DLR-A fuel stream is mixture of 22.1% CH_4, 33.2% H_2, and 44.7% N_2 (by volume) that issues into dry air via a nozzle with an inner diameter of 8 mm at a mean exit velocity of 42.2 $\mathrm{m{\cdot}s^{-1}}$. The coflow air stream issues from a concentric nozzle 140 mm in diameter at a velocity of 0.3 $\mathrm{m{\cdot}s^{-1}}$. The reported jet Reynolds number is 15,200.

Previous ODT ~~studies~~ simulations of turbulent jet flames have used the temporal planar formulation, but the spatial cylindrical formulation developed recently [30] more closely matches the experimental configuration. This simulation uses the experimentally reported velocity profiles and jet dimensions. The fuel was diluted with N_2 in the experimental flame to minimize radiative heat losses, and radiation is ignored in the simulation. This flame has a low Reynolds number, and the combustion chemistry proceeds quickly. The ODT simulation transports the chemical species O_2, N_2, CH_4, H_2, H_2O, and CO_2. We assume that reactions proceed to the products of complete combustion and apply simple, fast reaction rates according to the following chemical equations:

$$CH_4 + 2O_2 \rightarrow CO_2 + 2H_2O, \tag{2}$$

$$H_2 + \frac{1}{2}O_2 \rightarrow H_2O. \tag{3}$$

These assumptions are reasonable for illustrating ODT in the DLR-A reacting jet configuration with variable properties and heat release.

This simulation uses ODT parameters $C = 20$, $\beta_{LES} = 17$, and $Z = 400$. The values of C and β_{LES} were adjusted to give good agreement with the experimental data, and the value of Z is the same as the spatial simulations in [15]. 1024 independent flow realizations were performed in parallel and the results ensemble averaged. Downstream distance y and radial position r are normalized by the jet diameter D.

Figure 4 displays the simulation results: axial mean and centerline values for (a) axial velocity v, (b) mixture fraction ξ, and (c) temperature T. The ODT simulation results track the experimental data well for all three variables. The centerline temperature peaks about 100 K above the experimental data, but this small difference can be attributed to thermal radiation (which was neglected in this simulation) and the assumption that reactions proceed to the products of complete combustion rather than their equilibrium state. The centerline velocity shows a fast initial decrease that can be attributed to diffusion. Similarly, the increase in centerline velocity RMS values is delayed; this occurs due to the elapsed time model for large eddy suppression, which

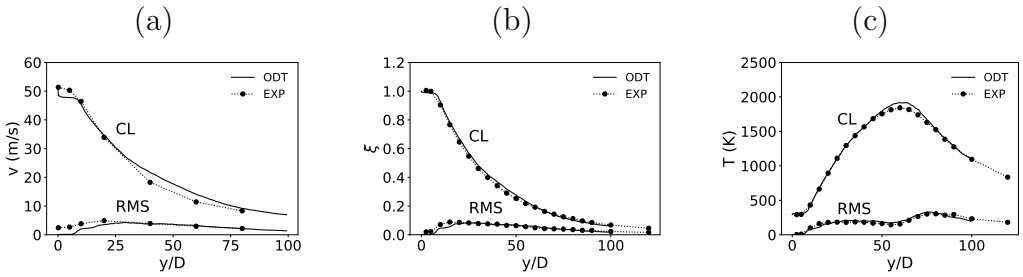

Figure 4: DLR jet flame example case results: (a) mean axial velocity and RMS velocity along the centerline versus downstream location; (b) mean axial mixture fraction and RMS mixture fraction along the centerline versus downstream location; (c) mean axial temperature (K) and RMS temperature (K) along the centerline versus downstream location.

limits disturbances in the early stages of the flow to small eddies that occur on the jet edges away from the centerline.

To replicate this example case, build the code with the `CHEMISTRY = SIMPLEDLR` flag in the `user_config` file and edit the desired run script with `inputDir = "../input/jetFlame/DLR_A"` and a new case name such as `caseName = "jetFlame_example"`. The input file for this case, located at `ODT/input/jetFlame/DLR_A/input.yaml`, contains the appropriate parameters and does not need to be modified to match this example case. See the code documentation for data post-processing instructions.

4. Impact

The ODT model is complex, challenging to implement, and difficult to cleanly separate from its implementation, all of which presents barriers to practitioners who wish to use the model. For instance, triplet mapa can be represented mathematically but are usually implemented as previously described, assuming an adaptive grid with variable cell sizes. Alternative formulations are used for fixed grids. The ~~code~~ ODT code implementation presented here reduces these barriers by providing an open-source, well-documented, modularly styled code that can be used directly or modified to suit particular research goals. Its structure facilitates addressing new research questions, such as soot-flame-radiation-chemistry interactions, multiphase reacting flows using Lagrangian particle models (such as diesel sprays), and coupled near-wall heat and momentum transfer. The code presented here has been extended to and demonstrated in such systems already, facilitated by the object-oriented structure. For example, this code has been extended to solve steady laminar flamelets [49], requiring inheritance from the `solver`,

`micromixer`, and `dv` (domain variable) classes.

The structure of the ODT code, using an adaptive mesh and a Lagrangian formulation for diffusive advancement of transport equations, has several advantages, including simplicity of formulation and computational efficiency. The adaptive mesh minimizes the number of computational cells required and can be adapted to any variable of interest. This is especially important for flows with complex and expensive chemistry and flows with widely varying resolution demands such as near-wall flows. The Lagrangian formulation implies that cell faces move with fluid velocity (such as is present during flame expansion, for example). This treatment is inherently stable, removing the need for complex and diffusive hyperbolic conservation law treatments and their associated CFL constraints. The conservation equations, as in Equation 1, are similarly simplified.

The code has been run on Linux, Mac, and Windows machines, including parallel clusters of up to 512 processor cores. Several research groups have used the code and contributed to its development. These include groups at Sandia National Laboratories, Brandenburg University of Technology Cottbus-Senftenberg, Chalmers University, and Combustion Science and Engineering, Inc. The latter company is the only organization known to use the code in a commercial setting, though on a combustion research project.

5. Conclusion

A clean and extensible implementation of the one-dimensional turbulence (ODT) model has been presented in an open-source, object-oriented C++ framework, motivated by advanced turbulent reacting flow research. The build process, directory structure, input file format, run process, and post-processing procedure were described in detail, including a representative reacting jet example case. Access to a proven, documented code and a modern build procedure will aid researchers in applying and extending the model to new research questions.

6. Conflict of Interest

There are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

Acknowledgements

Special thanks to Alan R. Kerstein for his significant contributions to model development. Additional thanks to Heiko Schmidt, Juan Medina, and Marten Klein of Brandenburg University of Technology Cottbus-Senftenberg, Michael Oevermann and Marco Fistler of Chalmers University of Technology, and John Hewson of Sandia National Laboratories for code contributions, and insightful conversations.

References

[1] D. O. Lignell, G. C. Fredline, A. D. Lewis, Comparison of one-dimensional turbulence and direct numerical simulations of soot formation and transport in a nonpremixed ethylene jet flame, Proceedings of the Combustion Institute 35 (2) (2015) 1199–1206. doi:10.1016/j.proci.2014.05.046.

[2] A. W. Abboud, C. Schulz, T. Saad, S. T. Smith, D. D. Harris, D. O. Lignell, A numerical comparison of precipitating turbulent flows between large-eddy simulation and one-dimensional turbulence, AIChE Journal 61 (10) (2015) 3185–3197. doi:10.1002/aic.14870.

[3] A. R. Kerstein, One-dimensional turbulence: model formulation and application to homogeneous turbulence, shear flows, and buoyant stratified flows, Journal of Fluid Mechanics 392 (1999) 277–334. doi:10.1017/S0022112099005376.

[4] A. R. Kerstein, T. D. Dreeben, Prediction of turbulent free shear flow statistics using a simple stochastic model, Physics of Fluids 12 (2) (2000) 418–424. doi:10.1063/1.870319.

[5] A. R. Kerstein, W. T. Ashurst, S. Wunsch, V. Nilsen, One-dimensional turbulence: vector formulation and application to free shear flows, Journal of Fluid Mechanics 447 (2001) 85–109. doi:10.1017/S0022112001005778.

[6] T. Echekki, A. R. Kerstein, T. D. Dreeben, J.-Y. Chen, 'one-dimensional turbulence' simulation of turbulent jet diffusion flames: model formulation and illustrative applications, Combustion and Flame 125 (3) (2001) 1083–1105. doi:10.1016/S0010-2180(01)00228-0.

[7] J. C. Hewson, A. R. Kerstein, Stochastic simulation of transport and chemical kinetics in turbulent $CH_4/H_2/N_2$ flames, Combustion Theory

399 and Modelling 5 (4) (2001) 669–697. doi:10.1088/1364-7830/5/4/
400 309.

401 [8] J. C. Hewson, A. R. Kerstein, Local extinction and reignition in non-
402 premixed turbulent $CH_4/H_2/N_2$ jet flames, Combustion Science and
403 Technology 174 (5-6) (2002) 35–66. doi:10.1080/713713031.

404 [9] D. O. Lignell, D. S. Rappleye, One-dimensional-turbulence simulation of
405 flame extinction and reignition in planar ethylene jet flames, Combustion
406 and Flame 159 (9) (2012) 2930–2943. doi:10.1016/j.combustflame.
407 2012.03.018.

408 [10] N. Punati, J. C. Sutherland, A. R. Kerstein, E. R. Hawkes, J. H. Chen,
409 An evaluation of the one-dimensional turbulence model: comparison
410 with direct numerical simulations of CO/H_2 jets with extinction and
411 reignition, Proceedings of the Combustion Institute 33 (1) (2011) 1515–
412 1522. doi:10.1016/j.proci.2010.06.127.

413 [11] A. Abdelsamie, D. O. Lignell, D. Thévenin, Comparison between ODT
414 and DNS for ignition occurrence in turbulent premixed jet combus-
415 tion: safety-relevant applications, Zeitschrift fur Physikalische Chemie
416 231 (10) (2017) 1709–1735. doi:10.1515/zpch-2016-0902.

417 [12] D. O. Lignell, V. B. Lansinger, A. R. Kerstein, A cylindrical formula-
418 tion of the one-dimensional turbulence (ODT) model for turbulent jet
419 flames, in: AIChE Annual Meeting 2017, American Institute of Chemi-
420 cal Engineers, 2017.

421 [13] B. Goshayeshi, J. C. Sutherland, Prediction of oxy-coal flame stand-
422 off using high-fidelity thermochemical models and the one-dimensional
423 turbulence model, Proceedings of the Combustion Institute 35 (3) (2015)
424 2829–2837. doi:10.1016/j.proci.2014.07.003.

425 [14] Z. Jozefik, A. R. Kerstein, H. Schmidt, S. Lyra, H. Kolla, J. H. Chen,
426 One-dimensional turbulence modeling of a turbulent counterflow flame
427 with comparison to DNS, Combustion and Flame 162 (8) (2015) 2999–
428 3015. doi:10.1016/j.combustflame.2015.05.010.

429 [15] E. I. Monson, D. O. Lignell, M. A. Finney, C. Werner, Z. Jozefik,
430 A. R. Kerstein, R. S. Hintze, Simulation of ethylene wall fires using
431 the spatially-evolving one-dimensional turbulence model, Fire Technol-
432 ogy 52 (1) (2016) 167–196. doi:10.1007/s10694-014-0441-2.

[16] J. C. Hewson, A. J. Ricks, S. R. Tieszen, A. R. Kerstein, R. O. Fox, Conditional-moment closure with differential diffusion for soot evolution in fire, in: Center for Turbulence Research, Proceedings of the Summer Program 2006, Stanford University, 2006.

[17] J. C. Hewson, A. J. Ricks, S. R. Tieszen, A. R. Kerstein, R. O. Fox, On the transport of soot relative to a flame: modeling differential diffusion for soot evolution in fire, in: H. Bockhorn, A. D'Anna, A. F. Sarofim, H. Wang (Eds.), Combustion Generated Fine Carbonaceous Particles, KIT Scientific Publishing, 2009, pp. 571–588.

[18] D. O. Lignell, J. C. Hewson, One-dimensional turbulence simulation: overview and application to soot formation in nonpremixed flames, in: SIAM Conference on Computational Science and Engineering, 2015.

[19] A. J. Ricks, J. C. Hewson, A. R. Kerstein, J. P. Gore, S. R. Tieszen, W. T. Ashurst, A spatially developing one-dimensional turbulence (ODT) study of soot and enthalpy evolution in meter-scale buoyant turbulent flames, Combustion Science and Technology 182 (1) (2010) 60–101. doi:10.1080/00102200903297003.

[20] G. Sun, J. C. Hewson, D. O. Lignell, Evaluation of stochastic particle dispersion modeling in turbulent round jets, International Journal of Multiphase Flow 89 (2017) 108–122. doi:10.1016/j.ijmultiphaseflow.2016.10.005.

[21] J. R. Schmidt, J. O. L. Wendt, A. R. Kerstein, Non-equilibrium wall deposition of inertial particles in turbulent flow, Journal of Statistical Physics 137 (2) (2009) 233–257. doi:10.1007/s10955-009-9844-8.

[22] G. Sun, D. O. Lignell, J. C. Hewson, C. R. Gin, Particle dispersion in homogeneous turbulence using the one-dimensional turbulence model, Physics of Fluids 26 (10) (2014) 103301. doi:10.1063/1.4896555.

[23] M. Fistler, D. O. Lignell, A. R. Kerstein, M. Oevermann, Numerical studies of turbulent particle-laden jets using spatial approach of one-dimensional turbulence, in: ILASS-Europe 28th Conference on Liquid Atomization and Spray Systems, 2017.

[24] S. Cao, T. Echekki, A low-dimensional stochastic closure model for combustion large-eddy simulation, Journal of Turbulence 9. doi:10.1080/14685240701790714.

[25] R. C. Schmidt, A. R. Kerstein, S. Wunsch, V. Nilsen, Near-wall LES closure based on one-dimensional turbulence modeling, Journal of Computational Physics 186 (1) (2003) 317–355. doi:10.1016/S0021-9991(03)00071-8.

[26] R. C. Schmidt, A. R. Kerstein, R. McDermott, ODTLES: A multi-scale model for 3D turbulent flow based on one-dimensional turbulence modeling, Computer Methods in Applied Mechanics and Engineering 199 (13-16) (2010) 865–880. doi:10.1016/j.cma.2008.05.028.

[27] E. Gonzalez-Juez, A. R. Kerstein, D. O. Lignell, Fluxes across double-diffusive interfaces: a one-dimensional-turbulence study, Journal of Fluid Mechanics 677 (2011) 218–254. doi:10.1017/jfm.2011.78.

[28] E. Gonzalez-Juez, A. R. Kerstein, D. O. Lignell, Reactive Rayleigh–Taylor turbulent mixing: a one-dimensional-turbulence study, Geophysical & Astrophysical Fluid Dynamics 107 (5) (2013) 506–525. doi:10.1080/03091929.2012.736504.

[29] S. Wunsch, A. R. Kerstein, A model for layer formation in stably stratified turbulence, Physics of Fluids 13 (3) (2001) 702–712. doi:10.1063/1.1344182.

[30] D. O. Lignell, V. B. Lansinger, J. Medina, M. Klein, A. R. Kerstein, H. Schmidt, M. Fistler, M. Oevermann, One-dimensional turbulence modeling for cylindrical and spherical flows: model formulation and application, Theoretical and Computational Fluid Dynamics 32 (4) (2018) 495–520. doi:10.1007/s00162-018-0465-1.

[31] M. Klein, D. O. Lignell, H. Schmidt, Map-based modeling of turbulent convection: application of the one-dimensional turbulence model to planar and spherical geometries, in: International Conference on Rayleigh-Benard Turbulence, 2018.

[32] M. Klein, D. O. Lignell, H. Schmidt, Stochastic modeling of temperature and velocity statistics in spherical-shell convection, in: EGU Conference on Recent developments in Geophysical Fluid Dynamics, 2019.

[33] W. T. Ashurst, A. R. Kerstein, One-dimensional turbulence: variable-density formulation and application to mixing layers, Physics of Fluids 17 (2). doi:10.1063/1.1847413.

[34] D. O. Lignell, A. R. Kerstein, G. Sun, E. I. Monson, Mesh adaption for efficient multiscale implementation of one-dimensional turbulence,

Theoretical and Computational Fluid Dynamics 27 (3-4) (2013) 273–295. doi:10.1007/s00162-012-0267-9.

[35] Y. A. Çengel, J. M. Cimbala, Fluid Mechanics, 2nd Edition, Çengel series in engineering thermal-fluid sciences, McGraw-Hill Higher Education, 2010.

[36] J. Beder, yaml-cpp v0.6.3 (2008).
URL https://github.com/jbeder/yaml-cpp/

[37] G. Strang, On the construction and comparison of difference schemes, SIAM Journal on Numerical Analysis 5 (3) (1968) 506–517. doi:10.1137/0705041.

[38] A. C. Hindmarsh, R. Serban, D. R. Reynolds, CVODE, https://computing.llnl.gov/sites/default/files/public/cv_guide.pdf (2020).
URL https://computing.llnl.gov/projects/sundials/cvode

[39] D. G. Goodwin, R. L. Speth, H. K. Moffat, B. W. Weber, Cantera (2018). doi:10.5281/zenodo.1174508.
URL https://cantera.org/

[40] D. van Heesch, Doxygen (2018).
URL https://www.doxygen.nl/

[41] A. B. Yoo, M. A. Jette, M. Grondona, Slurm: simple linux utility for resource management, in: D. G. Feitelson, L. Rudolph, U. Schwiegelshohn (Eds.), Job Scheduling Strategies For Parallel Processing, Vol. 2862 of Lecture notes in computer science, 0302-9743, Springer, 2003, pp. 44–60. doi:10.1007/10968987-3.

[42] W. Meier, R. S. Barlow, Y.-L. Chen, J.-Y. Chen, Raman/Rayleigh/LIF measurements in a turbulent $CH_4/H_2/N_2$ jet diffusion flame: experimental techniques and turbulence–chemistry interaction, Combustion and Flame 123 (3) (2000) 326–343. doi:10.1016/S0010-2180(00)00171-1.
URL https://tnfworkshop.org/data-archives/simplejet/dlrflames/

[43] H. Pitsch, Unsteady flamelet modeling of differential diffusion in turbulent jet diffusion flames, Combustion and Flame 123 (3) (2000) 358–374. doi:10.1016/S0010-2180(00)00135-8.

[44] R. P. Lindstedt, H. Ozarovsky, Joint scalar transported PDF modeling of nonpiloted turbulent diffusion flames, Combustion and Flame 143 (4) (2005) 471–490. doi:10.1016/j.combustflame.2005.08.030.

[45] H. Wang, S. B. Pope, Large eddy simulation/probability density function modeling of a turbulent $CH_4/H_2/N_2$ jet flame, Proceedings of the Combustion Institute 33 (1) (2011) 1319–1330. doi:10.1016/j.proci.2010.08.004.

[46] M. Fairweather, R. M. Woolley, First-order conditional moment closure modeling of turbulent, nonpremixed methane flames, Combustion and Flame 138 (1-2) (2004) 3–19. doi:10.1016/j.combustflame.2004.03.001.

[47] K. W. Lee, D. H. Choi, Prediction of NO in turbulent diffusion flames using eulerian particle flamelet model, Combustion Theory and Modelling 12 (5) (2008) 905–927. doi:10.1080/13647830802094351.

[48] K. W. Lee, D. H. Choi, Analysis of NO formation in high temperature diluted air combustion in a coaxial jet flame using an unsteady flamelet model, International Journal of Heat and Mass Transfer 52 (5-6) (2009) 1412–1420. doi:10.1016/j.ijheatmasstransfer.2008.08.015.

[49] N. Peters, Laminar diffusion flamelet models in non-premixed turbulent combustion, Progress in Energy and Combustion Science 10 (1984) 319–339.

Current executable software version

Software information is provided in Table 2.

Nr.	(Executable) software metadata description	Please fill in this column
S1	Current software version	1.0
S2	Permanent link to executables of this version	For example: https://github.com/BYUignite/ODT
S3	Legal Software License	MIT
S4	Computing platforms/Operating Systems	Linux, OS X, Microsoft Windows
S5	Installation requirements & dependencies	CMake 3.12+, Cantera, Git, Doxygen (optional)
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	https://github.com/BYUignite/ODT
S7	Support email for questions	davidlignell@byu.edu

Table 2: Software metadata (optional)