RESEARCH ARTICLE

Studying Lagrangian dynamics of turbulence using on-demand fluid particle tracking in a public turbulence database

Huidan Yu[a,†], Kalin Kanov[b], Eric Perlman[b], Jason Graham[a], Edo Frederix[a,+], Randal Burns[b], Alexander Szalay[c], Gregory Eyink[d] and Charles Meneveau[a*]

[a]*Department of Mechanical Engineering*; [b]*Department of Computer Science*; [c]*Department of Physics and Astronomy*; [d]*Department of Applied Mathematics and Statistics, The Johns Hopkins University, 3400 North Charles Street, Baltimore, MD 21218, USA*; [†]*present address: Mechanical Engineering, Indiana University-Purdue University Indianapolis*;

(*v3.2 released February 2009*)

A recently developed JHU public turbulence database [1, 2] provides new ways to access large datasets generated from high-performance computer simulations of turbulent flows to perform numerical experiments. The database archives 1024^4 (spatial & time) data points obtained from a pseudo-spectral direct numerical simulation (DNS) of forced isotropic turbulence. The flow's Taylor-scale Reynolds number is $Re_\lambda = 443$, and the simulation output spans about one large-scale eddy turnover time. Besides the stored velocity and pressure fields, built-in 1st- and 2nd-order space differentiation as well as spatial and temporal interpolations are implemented on the database. The resulting 27 terabytes (TB) of data are public and can be accessed remotely through an interface based on a modern Web-services model. Users may write and execute analysis programs on their host computers, while the programs make subroutine-like calls (*getFunction*s) requesting desired variables (velocity and pressure and their gradients) over the network. The architecture of the database and the initial built-in functionalities are described in a previous JoT paper [2]. In the present paper, further developments of the database system are described; mainly the newly developed *getPosition* function. Given an initial position, integration time-step, as well as an initial and end time, the *getPosition* function tracks arrays of fluid particles and returns particle locations at the end of the trajectory integration time. The *getPosition* function is tested by comparing with trajectories computed outside of the database. It is then applied to study Lagrangian velocity structure functions as well as tensor-based Lagrangian time correlation functions. The roles of pressure Hessian and viscous terms in the evolution of the symmetric and antisymmetric parts of the velocity gradient tensor are explored by comparing the time correlations with and without these terms. Besides the *getPosition* function, several other updates to the database are described such as a function to access the forcing term in the DNS, a new more efficient interpolation algorithm based on partial sums, and a new Matlab interface.

Keywords: Forced isotropic turbulence; Lagrangian time correlation; Particle tracking; Turbulence database; Web services.

1. Introduction

Due to advances in computer hardware and algorithms, turbulence simulations supported by high performance computing infrastructures have continued to expand. Direct Numerical Simulations (DNS) of turbulent flows using on the order of $1000^3 - 4000^3$ grid points have been reported [3–6]. In the turbulence research

*Corresponding author. Email: meneveau@jhu.edu, [+]permanent address: Mechanical Engineering, Eindhoven University of Technology, The Netherlands

community, the prevailing approach is that individual researchers perform large simulations that are analyzed during the runs and only a small subset of time-steps are stored for subsequent, and by necessity more "static", analysis. A number of representative snapshots are stored while the majority of the time evolution has to be discarded. As a result, much of the computational effort is not utilized as effectively as it could. In fact, often large simulations of the same process must be repeated after new questions arise that were not initially obvious. Storing the entire space-time history of a simulation, however, generates datasets that are very large and very difficult to access using prevailing approaches. Thus, the increasingly larger, top-ranked simulations run the risk of becoming less and less accessible to the wider turbulence scientific community.

As a step to develop new effective ways to translate the massive amounts of computational turbulence data into meaningful knowledge, a new "cyber fluid dynamics" paradigm has been proposed, which combines high-fidelity DNS of turbulence with modern database technology [2]. The newly created JHU public turbulence database (*http://turbulence.pha.jhu.edu*) archives a 27 Terabytes (TB) dataset from a direct numerical simulation (DNS) of forced isotropic turbulence consisting of 1024^4 (spatial and time) samples, spanning about one large-scale eddy turnover time. The database stores velocity and pressure fields. The domain size is in a $[0, 2\pi]^3$ domain and the Taylor-microscale Reynolds number is $Re_\lambda \simeq 433$. The spatial resolution is $dx = 2\pi/1024$ and the Kolmogorov scale is $\eta_K = 0.00287$ so that $dx/\eta_K \sim 2.1$. The turbulence integral scale is $L = 1.376$, the velocity root-mean-square value is $u' = 0.681$ and the mean dissipation-rate is $\epsilon = 0.092$, in the units of the simulation. The Kolmogorov time scale is $\tau_K = 0.045$. The stored time steps are separated by a time-interval of 0.002 (the original DNS was performed with a time-step of 2×10^{-4} using a very conservative CFL condition [2]).

One of the hallmarks of the database is a Web services interface that allows users to access data in a user-friendly fashion while allowing maximum flexibility to execute desired analysis tasks. Remote users may write and execute analysis programs on their own computers, while their programs make subroutine-like calls named *get-functions* (e.g. *getVelocityAndPressure*, *getVelocityGradient*, *getPressureLaplacian*, etc.) requesting desired variables such as velocity, pressure and their gradients, over the network. First and second-order space differentiation as well as spatial and temporal interpolations are implemented on the database as pre-defined functions. Instead of being restricted to analysis on the fly during DNS, researchers may write and execute more specialized analysis programs on their host computers at any time.

The data and the initial built-in functionalities have already been described in detail in a previous publication [2]. Due to the easy accessibility and flexibility, the database has attracted researchers from all over the world since its inception and its use has resulted in various publications [7–15]. Nevertheless, current functionalities focus on extracting data at single time-steps of the turbulent field, best suited for Eulerian studies of turbulence. There is also considerable interest in Lagrangian descriptions of turbulence. A Lagrangian description of turbulence has advantages in studies of turbulent transport and mixing processes, as well as relating statistics with dynamical descriptions following fluid particles. An extensive database of pre-computed Lagrangian trajectories for a large number of fluid and inertial particles, and turbulence quantities along the trajectories, has been in operation for several years [16].

The study of turbulence from a Lagrangian viewpoint has a long history, with the earliest works of Taylor [17] and Richardson [18] both pre-dating Kolmogorov [19]. It was recognized that transport issues are addressed naturally from the Lagrangian

viewpoint, which has since been successfully employed in the theoretical treatment of turbulent mixing [20–24]. Lagrangian concepts are also useful when considering entrainment processes at turbulent/laminar interfaces [25] and Lagrangian stochastic models are widely used to model processes ranging from atmospheric pollution transport to turbulent combustion [26]. Lagrangian dynamics of the velocity gradient tensor can be used to understand many fundamental and intrinsic properties of small-scale motions in high-Reynolds turbulence [13]. Studying Lagrangian turbulence requires following a large number of particle trajectories in order to capture the overall space and time scales. In spite of significant progress in recent years [23, 27, 28], experimental Lagrangian measurements remain challenging especially for high Re turbulence. Extraction of Lagrangian data from DNS is conceptually easy, but requires the full time evolution to have been stored, such as in the JHU turbulence database, or to store the trajectories of a predefined set of particles [16]. To track fluid particles with arbitrary initial locations, or even for backward tracking over extended time periods, trajectories must be recomputed on demand. However, using currently available tools in the JHU turbulence database, users must send requests back and forth over the network for each integration time step of the particle tracking. Improvements to this approach must follow the best practices of databases, such as "move the program to the data" [29].

As a new tool to facilitate Lagrangian analysis, we develop the *getPosition* function inside the database. It tracks arrays of particles moving with the flow and returns particle locations at the end of the trajectory integration time. The relevant algorithm and data management approach for the new *getPosition* function is described in §2, and the implementation is tested by comparing trajectories computed inside and outside the database. In Section 3, we study Lagrangian velocity structure functions and compare the results from the JHU database at $R_\lambda \sim 430$ to results from the literature at other Reynolds numbers. In Section 4, we study Lagrangian time correlation functions of the symmetric and antisymmetric parts of the velocity gradient tensor in which the impact of various terms in the corresponding dynamical evolution equation is quantified by systematically including these terms separately. In Section 5, we use the data to examine important features of a model for the pressure Hessian tensor and how its predictions compare with the data. We summarize the results in Section 6 with a short discussion. Other recent updates to the JHU database, such as the new *getForce* function, more efficient algorithms for interpolation, as well as new Matlab interfaces, are presented and documented in Appendices A, B and C, respectively.

2.   The *getPosition* function: algorithm and data handling

Existing database built-in functionalities can retrieve velocity, pressure as well as their derivatives at a specific location and time within the archived time history. To study Lagrangian turbulence one needs to perform an integration operation along fluid particle trajectories, e.g. using a Runge-Kutta method, which at present requires data transfers between a user's computer and the database at every small time-step needed in the Lagrangian integration. A more user-friendly and efficient approach would be for a user to let the database compute the fluid trajectories by doing the computations in the database. In this section we describe the algorithm used for such integration, as well as the way the computations are performed inside the various database layers. The end result is a new *getfunction* called *getPosition* that allows to track arrays of fluid particles simultaneously and returns final particle locations at the end of the specified trajectory integration time. It supports both forward and backward tracking of fluid particles.

2.1. *Fluid particle tracking algorithm*

The *getPosition* function uses second order accurate Runge-Kutta integration. Given fluid particle locations $\mathbf{X}_p$ at a user specified start time (t_{ST}), the function returns the particle locations at a user specified end time (t_{ET}). The user also specifies a particle integration time-step (Δt_p^*). Forward tracking is accomplished by specifying $t_{ET} > t_{ST}$, whereas backward tracking is accomplished by specifying $t_{ET} < t_{ST}$. The sign of the time-step need not be specified to make the distinction between forward and backward tracking since inside the tracking algorithm, it is taken to be $\Delta t_p = \text{sign}[t_{ET} - t_{ST}]|\Delta t_p^*|$.

Particle tracking is accomplished by integrating the following equation between times t_{ST} and t_{ET}

$$\frac{d\mathbf{x}_p}{dt} = \mathbf{u}(\mathbf{x}_p, t), \quad \mathbf{x}_p(t_{ST}) = \mathbf{X}_p, \tag{1}$$

where $\mathbf{x}_p(t)$ and $\mathbf{u}(\mathbf{x}_p, t)$ denote the position of the fluid particle originating (at initial time t_{ST}) from position $\mathbf{X}_p$ and the velocity field at the particle location, respectively. To advance the particle positions between two successive time instants t_m and $t_{m+1}(= t_m + \Delta t_p)$, the predictor step yields an estimate

$$\mathbf{x}_p^*(t_m) = \mathbf{x}_p(t_m) + \mathbf{u}(\mathbf{x}_p(t_m), t_m)\Delta t_p. \tag{2}$$

The corrector step then gives the particle position at t_{m+1} as

$$\mathbf{x}_p(t_{m+1}) = \mathbf{x}_p(t_m) + \Delta t_p \frac{1}{2}\left[\mathbf{u}(\mathbf{x}_p(t_m), t_m) + \mathbf{u}(\mathbf{x}_p^*(t_m), t_{m+1})\right]. \tag{3}$$

or

$$\mathbf{x}_p(t_{m+1}) = \mathbf{x}_p(t_m) + \frac{1}{2}\left[\mathbf{x}_p^*(t_m) - \mathbf{x}_p(t_m)\right] + \frac{1}{2}\Delta t_p\mathbf{u}(\mathbf{x}_p^*(t_m), t_{m+1}). \tag{4}$$

The integration proceeds until t_m reaches the user-specified final time t_{ET}. The last integration time-step is typically done using a smaller time-step so that the integration ends exactly at the specified t_{ET}. GetPosition then returns $\mathbf{x}_p(t_{ET})$ for all particles that were at initial locations $\mathbf{X}_p$.

For this integration scheme, the time-stepping error is of order $(\Delta t_p)^3$ over one time step. In general, accurate spatial and time interpolations are crucial to obtain the fluid velocities while tracking particles along their trajectories. Spatial interpolation with various optional orders of accuracy can be specified by the user. Time interpolation is done by default using PHCIP [2]. To call this function, a user needs to provide the start time (t_{ST}), particle number, an array containing the positions of each particle at t_{ST}, the integration time step Δt_p^*, and the end time (t_{ET}). On output, an array containing the positions of each particle at end time t_{ET} is returned. As a time-step for particle tracking, in what follows we use $\Delta t_p = 0.0004$ for tests and applications (i.e. there are five particle-tracking time-steps for each database time-step), unless stated otherwise.

2.2. *Data and particle movements across servers*

Due to the movement of particles within different portions of the data volume, the implementation of the *getPosition* function on the various database layers is less straightforward than the existing functions. In general, the data sets are stored in

multiple data servers, e.g. the current 27TB DNS data are partitioned across six data servers. For the existing functions, since only one specific time is touched, all the operations associated with a particular point in space, including temporal and spatial interpolation, differentiation, etc. can be executed within one of the data servers that store the data. The upper level web server plays a role to break down a user's batch query into pieces corresponding to each of the database servers and thus assigns each point query to a particular server, where the data retrieval and computation happens. Each of the data servers perform the requested computation for their portion of the entire batch. The retrieved variables are returned to the web server, where they are assembled and sent back to the user.

For *getPosition*, due to the movement of the fluid particles, it is often the case within the desired time integration period, that particles leave one data server and enter another data server either after the prediction semi-step, Eq. (2) or the full time step, Eq. (4). In our current implementation, in order to alleviate the individual database servers from the burden of keeping track of each individual particle and whether it is within the boundaries of each server, we reassign all of the particles after each semi or full step.

Figure 1 shows the movement of data between the web server and the database servers. During the first iteration of the algorithm the predictor step is evaluated. The initial set of particle positions $(\mathbf{x}_p(t_m))$ is distributed among the database servers according to the spatial and temporal partitioning of the data. This step requires the velocity for each particle at the initial positions $(\mathbf{u}(\mathbf{x}_p(t_m), t_m))$. The distribution of points across database servers ensures that the data are available locally on each database server. Each set of predictor positions $(\mathbf{x}_p^*(t_m))$ is evaluated according to Eq. (2) in the computational module of each database server. The predictor positions are then returned to the web server. Using the predictor positions the web server reassigns the particles to the database servers, and the corrector step is evaluated using Eq. (4). This step requires the retrieval of the velocity for each particle at the predictor positions $(\mathbf{u}(\mathbf{x}_p^*(t_m), t_{m+1}))$ and the initial particle positions, both of which are provided by the web server. Positions $\mathbf{x}_p(t_{m+1})$ are again evaluated in the computational module of each database server and returned to the web server. This process continues until the specified end time t_{ET} is reached. The reassignment of particles before each step in the Runge-Kutta integration ensures that the data requested for each particle position is guaranteed to be found on the database server that is issuing the request and performing the integration.

2.3. *Accuracy tests and performance*

The accuracy of particle tracking inside the database (using *getPosition*) is tested by comparing the trajectories with those evaluated using particle tracking as coded on a local host (called "local tracking"), which involves calls to the database at each of the integration time-steps. The integration algorithm in both methods is identical, as described above. It is found that both approaches return the same trajectories, typically up to the 6th or 7th digit after the decimal point (essentially machine accuracy and chaotic behavior). The agreement is illustrated below by comparing *getPosition* and local coding for two fluid particles and tracking them from the beginning to final time available in the database. Fig. 2 shows the coordinates of two particles starting from $x = 3.02$, $y = 3.57$, $z = 5.36$ (empty symbols) and $x = 3.97$, $y = 4.96$, $z = 4.29$ (solid symbols,) and moving along with the local flow. The particles are tracked in the whole time domain until $t \approx 45\tau_K$. Solid lines denote the integration done on a "local computer", whereas symbols denote the
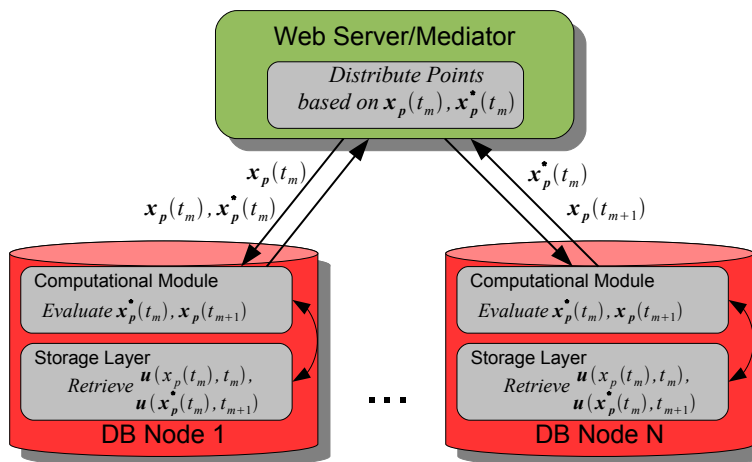
Figure 1.   Interaction and data movement between the web server and database server during the execution of the *getPosition* function.
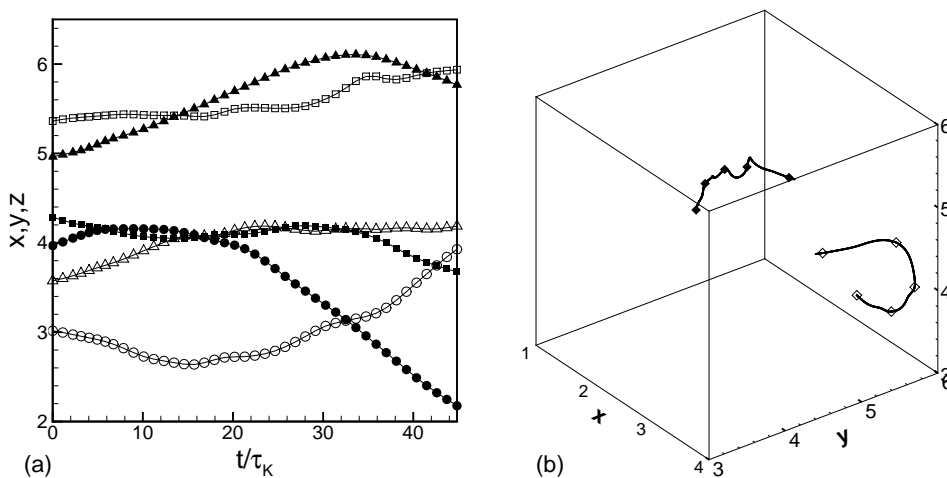
integration done using the *getPosition* function.

Figure 2.   (a) Time evolution of 2 arbitrary fluid particles in x,y,z directions using *getPosition* function (symbols) and local coding (lines) using the same integral algorithm. Circles: x; triangles: y, squares: z. Empty symbols: particle 1 starting from $x = 3.02$, $y = 3.57$, $z = 5.36$; solid symbols: particle 2 starting from $x = 3.97$, $y = 4.96$, $z = 4.29$. Lines denote the integration done on a "local computer", whereas symbols denote the integration done using the *getPosition* function. (b) 3D view of 2 particle tracks (symbols same as in part (a))

A noticeable feature of *getPosition* compared to the Eulerian-based *getFunctions* is the time expense due to the needed small integration time-step and the need to call the *getVelocity* function twice in each integration step, as described in Section 2 Eqs. 1-4. For large number of particles (e.g. over 100) and long integration time, the resulting calls can be very time-consuming if the particles are selected randomly in the entire domain. In practice, it is more efficient to collect particles from several randomly selected sub-cubes, e.g. consisting of 16^3 or 32^3 DNS grid-points. This is more efficient because it minimizes I/O of the data that are stored in atoms of size 72^3 [2]. Overall, more sampling particles are typically needed to achieve the same statistical convergence as compared to sampling randomly over the entire domain, but the overall efficiency is still significantly improved with such "sub-cube sampling".

When comparing the speed of the *GetPosition* function with the speed of tracking the particles on a local computer, we remark that it is difficult to obtain fully

repeatable performance measures, since the performance depends greatly on typical network speeds and system load, which can vary greatly over time. Nevertheless, the relative trends as shown in Fig. 3 are typically observed. The figure compares integration times when using *GetPosition* and particle tracking coded on a local computer ("local coding"), respectively, using exactly the same algorithm. To perform the comparison, we select ten sub-cubes of size N (N gridpoints on a side) at random locations, and track N^3 particles starting at each of the grid-points inside each sub-cube. We test two integration time-steps Δt_p and total tracking time $t_{ET} - t_{ST}$. The time required to finish the total integration is obtained for each of the 10 sub-cubes, and the times are averaged over the 10 sub-cubes. Sub-cube sizes between $N = 2$ and $N = 49$ were used, corresponding 8 to 117,649 particles being tracked in each cube. In Fig. 3, *getPosition* clearly shows the speedup against the local coding. For larger Δt_p and small $t_{ET} - t_{ST}$ in plot (a), the speedup increases suddenly as the particle number increases above $\sim 4,000$. For large particle numbers, the time expense to use *getPosition* function can be 3 times less than tracking the particles on a local computer relying on data transfers at each intermediate time. For the case of smaller Δt_p and longer $t_{ET} - t_{ST}$ shown in plot (b), the execution time is observed to increase more gradually. The time required to finish the integration time needed for *GetPosition* increases weakly even as the particle number increased significantly. This is because the sub-cube size (N) is always smaller than the 72^3 data-atoms. Hence, I/O needs are taxed about the same regardless of the value of N.
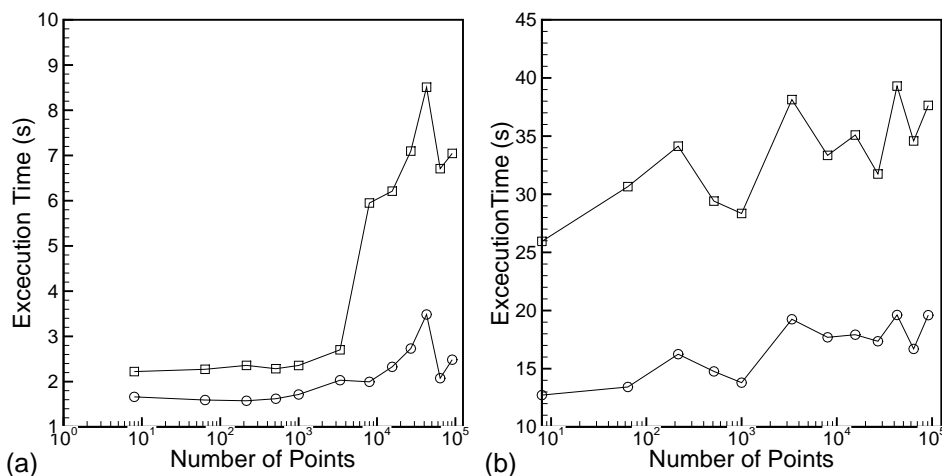

Figure 3. Performance comparison of *getPosition* (circles) vs. local coding (squares) of integration for particle tracking with integration time-step Δt_p and time $t = t_{ET} - t_{ST}$. (a) $\Delta t_p = 6.67 \times 10^{-4}$, $t_{ET} - t_{ST} = 6.0 \times 10^{-3}$; (b) $\Delta t_p = 4.0 \times 10^{-4}$, $t_{ET} - t_{ST} = 2.0 \times 10^{-2}$.

As mentioned before, the *getPosition* function may be used for backward tracking. An interesting test is to track particles forward in time, arrive at some final position, and then follow this operation by backward tracking for the same amount of time, in order to determine how far from the original position the fluid particle has been displaced. In the absence of roundoff and discretization errors, one would expect the initial and final positions to be the same independent of time. In the presence of roundoff and discretization errors in a highly chaotic flow, one expects exponential spreading of fluid particle displacements, with Lyapunov exponents on the order of the appropriate inverse eddy turnover time-scales. Suppose a fluid particle is located at $\mathbf{X}_p$ initially. It is first tracked forward from the initial time $t_{ST} = 0$ until a time $t_{ET} = t$ using *getPosition* (with an integration time-step

$\Delta t_p = 0.0004$). The final positions are then used as initial positions for backward tracking, setting $t_{ST} = t$ and $t_{ET} = 0$ (with the same integration time-step $|\Delta t_p| = 0.0004$). We denote the return location as $\mathbf{X}'_p$. Such tracking is performed for a set of thousands of particles. For small t we track 10,000 particles, for intermediate t we use 6,000 particles, while for long times ($t > 1.7$), a set of 2,000 fluid particles is tracked. The difference of $\mathbf{X}_p$ and $\mathbf{X}'_p$ is quantified using the *root-mean-square* position-difference (denoted as δx_{rms}, δy_{rms} and δz_{rms}) of the three components of the position-difference vector $\mathbf{X}'_p - \mathbf{X}_p$.

The results from such forward and backward tracking tests are shown in Fig. 4 where the three rms values are shown as function of the forward-backward integration time t. At small t, the errors observed in Fig. 4 are on the order of machine accuracy 10^{-7} and can thus be considered to be round-off errors. At larger t, the growth of rms displacement appears consistent with Lyapunov exponents appropriate for the different separation scales. At small t (at $t < \tau_K$), we expect that the errors will grow as $\sim \exp(t/\tau_K)$, where $\tau_K = 0.044$ is the Kolmogorov time-scale of the data. This translates into a relatively steep slope of $log_{10}(e)/0.044 \sim 10$. Up to times $t \approx 1$ the rms separation distance upon return remains smaller than one grid-spacing $dx \sim 6 \times 10^{-3}$. Once the separation distances grow to scales pertaining to the inertial range, one expects Lyapunov exponents on the order of $\epsilon^{1/3} \delta x_{\mathrm{rms}}^{-2/3}$ (where $\epsilon \approx 0.093$ is the mean dissipation rate of the data). For example, for $\delta x_{\mathrm{rms}} = 0.1$, this corresponds to a slope of $log_{10}(e) \times 0.093^{1/3} \times 0.1^{-2/3} \sim 0.9$ in the log-linear plot. The range of slopes mentioned is quite consistent with the trends observed in Fig. 4.
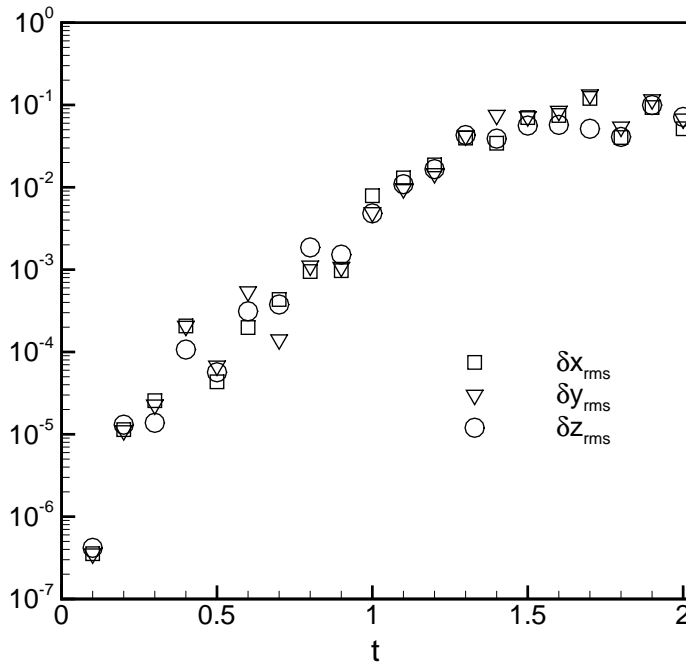

Figure 4.   Root mean square of the three coordinates of the position-difference vector arising from forward and backward fluid particle tracking using the *getPosition* function, plotted as function of forward-backward tracking time t.

3.   Lagrangian velocity structure functions

In this section, the new *getPosition* function is used to evaluate Lagrangian structure functions. It is well known that velocity differences across two points separated by a distance r are highly intermittent in the inertial range of scales for $\eta \ll r \ll L$ [30]. Much of the past evidence for intermittency has been obtained from Eulerian quantities, i.e. the moments of the spatial velocity increments. Among others, anomalous scaling of velocity increment moments, and the evolving shapes of their probability density functions (PDF) at different scales, are regarded as Eulerian hallmarks of intermittency [30]. Intermittency in temporal velocity statistics, which for proper Galilean invariance properties should be evaluated in a Lagrangian frame, moving with the fluid, has been studied in detail only more recently. This is due to advances in experimental techniques [23, 27, 28] and in computer simulations [31], as well as the availability of Lagrangian time-series of turbulence (such as that from data described in [32]).

A quantity of central interest for Lagrangian studies of turbulence is the Lagrangian velocity structure function (LVSF). In analogy to the Eulerian velocity structure function, the LVSF is defined as

$$S_p(\tau) = <(\delta_\tau v)^p> = <[v(t+\tau) - v(t)]^p> \tag{5}$$

where v denotes a single velocity component of a fluid particle. The time-lag is taken along a fluid particle trajectory. There have been detailed assessments of the scaling behavior, $S_p(\tau) = <(\delta_\tau v)^p> \sim \tau^{\xi(p)}$, with a focus on the scaling exponent $\xi(p)$ and its dependence on moment order p. Recently, Biferale *et al* [31] presented a detailed comparison between state-of-the-art experimental and numerical data of LVSF in turbulence. In their paper [31], the DNS data were obtained from a statistically homogeneous and isotropic turbulent flow with $Re_\lambda = 178$ and 284. The experimental data were obtained at Reynolds number ranging from $Re_\lambda = 350$ to $Re_\lambda = 815$ in a swirling water flow between counter-rotating baffled disks. They analyze intermittency at both short, $\tau \approx \tau_\eta$, and intermediate $\tau_\eta < \tau \ll T_L$, time lags.

Here we use the DNS data ($Re_\lambda = 443$) in the JHU turbulence database to compute the LVSFs. We use the *getPosition* function to track about $14,000$ fluid particle trajectories. For the sake of efficiency when calling *getPosition* as mentioned above, we collect the particle trajectories starting from sub-cubes chosen randomly from the entire domain, and then randomly select particles in each sub-cube. The particle number varies with the sub-cube size, as indicated in Table 1. The largest time lag is about $45\tau_K$, within the database's available time range.

Sub-cube size	Number of Sub-cubes	Particles per sub-cube	Total particles #
128	2	800	1600
64	20	400	8000
32	26	200	3200
16	15	100	1500

Table 1.   Sub-cube sizes and particle numbers used for starting location of fluid particle tracking.

Figure 5 shows a compilation of normalized second-order LSVFs at different Reynolds numbers, from three datasets from DNS and four datasets from experimental measurements. The solid line is from the analysis of the data in the JHU turbulence database, and symbols are reproduced from Fig. 1 in Ref. [31]. Solid symbols are for two DNS results with relatively low Re numbers and empty symbols are for experimental data. The second-order LVSF increases in a short time
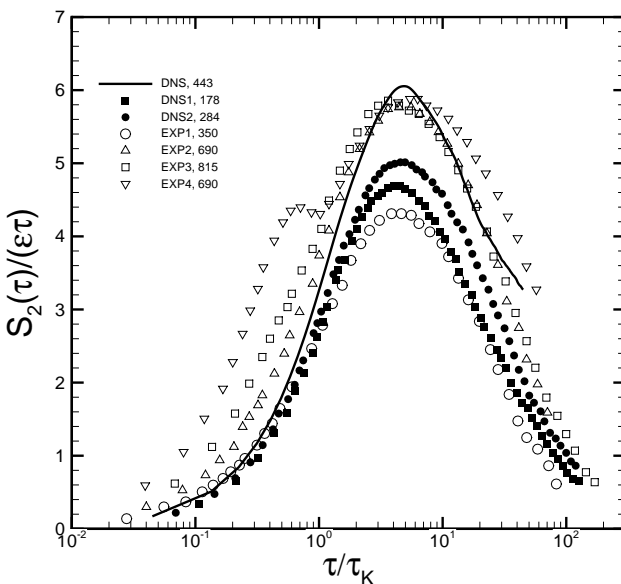
Figure 5. Time evolution of normalized second-order Lagrangian velocity structure function (averaged over the three components), i.e. $S_2(\tau)/(\varepsilon\tau)$, at various Reynolds numbers in turbulence. The solid line is computed from the data in the JHU turbulence database ($R_\lambda = 443$), solid symbols and empty symbols reproduced from Fig. 1 in Ref. [31].

range ($\tau < \tau_K$), reaches a maximum at $\tau \approx 5\tau_K$, and then decreases at large times, ($\tau > 10\tau_K$). However, no extended plateau is observed in the intermediate time range, indicating that the power law regime typical of the inertial range has not yet been achieved. The trends are mostly consistent between low Re DNS and high Re experiments, although near the peak, the present results overshoot the experimental data by about 3%.

Based on the standard Kolmogorov scaling that assumes $S_p(\tau) \propto v_{rms}^p Re_\lambda^{-p/2}(\tau/\tau_K)^{p/2}$ where the relations of $\varepsilon \propto v_{rms}^3/L$ and $T_L/\tau_K \propto Re_\lambda$ have been used (see [31]), we plot the second- and fourth-order LVSF compensated using $Re_\lambda^{p/2}/v_{rms}^p$ in Figs. 6. Again, the solid line is from the JHU turbulence database and symbols are digitized from Fig. 3 in Ref. [31]. It is seen that the solid line follows the trends of the other data sets quite well, with good collapse between the various lines for different Reynolds numbers as indicated.

It is concluded that the *getPosition* function can be used quite effectively to probe Lagrangian statistics in turbulence. There is good agreement with prior data regarding temporal, Lagrangian structure functions.

4. Tensor-based Lagrangian time-correlations of strain and rotation rates

The dynamics of the velocity gradient tensor A_{ij} is of significant interest [13] because it encodes rich information about turbulence through its nine components (in three dimensions). The Lagrangian autocorrelation time-scales for the tensor elements themselves [8, 9, 33] are of particular interest in the construction of models and for general physical understanding. The time evolution of **A** following fluid particles can be obtained quite simply by taking the gradient of the NS equations. For incompressible flow, the resulting equation reads

$$\frac{dA_{ij}}{dt} = -A_{ik}A_{kj} - \frac{\partial^2 p}{\partial x_i \partial x_j} + \nu \frac{\partial^2 A_{ij}}{\partial x_k \partial x_k} \tag{6}$$
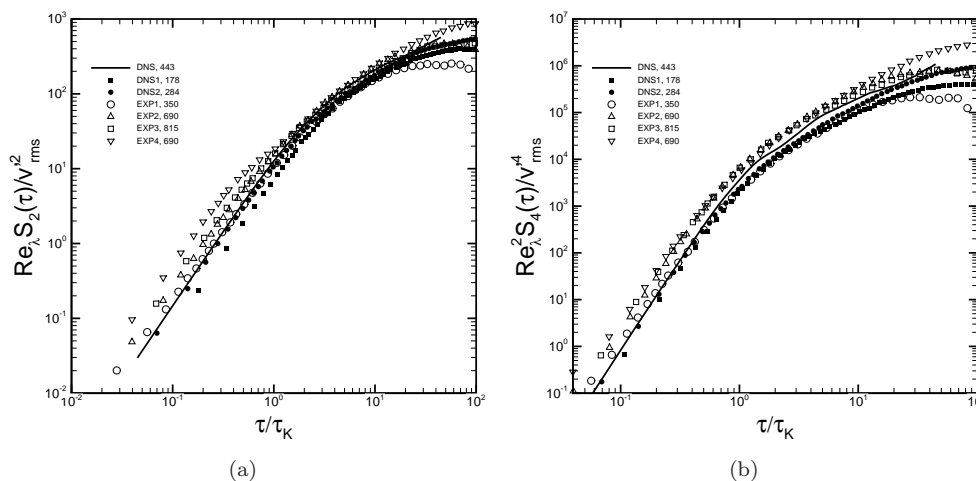
Figure 6.   Log-log plots of the second- and forth-order LVSF compensated using $Re_\lambda^{p/2}/v_{rms}^p$ vs. normalized time lag. (a). $p = 2$; (b). $p = 4$. The solid line is computed from JHU turbulence database $(R_\lambda = 443)$, solid symbols and empty symbols are obtained from Fig. 1 in Ref. [31].

where d/dt stands for Lagrangian material derivative and p is the pressure divided by the density of the fluid. The first term on the right-hand side of Eq. (6) denotes the nonlinear self-interaction of $\mathbf{A}$, the second term is a tensor called pressure Hessian $P_{ij} \equiv \partial^2 p/\partial x_i \partial x_j$, and the third is the viscous term. The tensor $\mathbf{A}$ contains 9 elements among which 8 components are independent noticing the incompressibility condition. The elements by themselves (unlike e.g. S^2 and Ω^2), are not directly coordinate system invariant. So rather than evaluating 9 separate temporal autocorrelation functions for each tensor element, their characterization should be done more compactly in a frame-invariant fashion. With this in mind, as in Refs. [8, 9] we use the tensor-based Lagrangian time correlation function of a second-rank tensor $\mathbf{C}$ defined as

$$\rho_{\mathbf{C}}(\tau) \equiv \frac{\langle C_{ij}(t_0) C_{ij}(t_0 + \tau) \rangle}{\sqrt{\langle (C_{mn}(t_0))^2 \rangle \cdot \langle (C_{pq}(t_0 + \tau))^2 \rangle}}, \tag{7}$$

The tensor $\mathbf{C}$ can be taken as $\mathbf{A}$ but the same can also be done for the strain-rate tensor $\mathbf{S} = \frac{1}{2}(\mathbf{A}+\mathbf{A}^T)$ and the rotation tensor $\mathbf{\Omega} = \frac{1}{2}(\mathbf{A}-\mathbf{A}^T)$, as well as the pressure Hessian tensor P_{ij} in Eq. (6).

Measurements are presented of $\rho_{\mathbf{C}}(\tau)$ for $C_{ij} = A_{ij}$, S_{ij}, and Ω_{ij} using the JHU turbulence database with the Lagrangian tracking done using the *getPosition* function. Over 10,000 fluid particles initially located at random positions in the whole domain are tracked over one large eddy turnover time. Tensor values are extracted along their trajectories. We show results on tensor-based Lagrangian auto-correlation functions of velocity gradient $\mathbf{A}$ and its related parts in Fig. 7. Repeating the analysis already performed in Ref. [8] we present also the auto-correlation functions of $\mathbf{S}$ (dash line) and $\mathbf{\Omega}$ (dash dot line). As shown in Fig. 7 and discussed in Ref. [8], the rotation exhibits much more "persistence" in time than deformation-rate. As can be expected, $\rho_{\mathbf{A}}$ (sold line in Fig. 7) falls between $\rho_{\mathbf{S}}$(dash line) and $\rho_{\mathbf{\Omega}}$(dash dot line) because $\mathbf{A}$ is a linear combination of $\mathbf{S}$ and $\mathbf{\Omega}$. Quantitatively, as $\mathbf{S}$ completely loses its time memory when the time lag is long enough ($\tau \sim 10\tau_K$), correlation functions of $\mathbf{A}$ and $\mathbf{\Omega}$ display slower decay and lose most correlation after about $\tau \sim 30\tau_K$. These trends were already noted in Ref. [8]. While qualitatively this result is in a good agreement with theoretical predictions [34], numerical simulations [35, 36] and experimental observations [37],

here the difference between strain- rate and rotation is much more marked than that implied by the prior studies which focused on the scalar square-magnitudes of these variables.

A natural question to ask is what are possible factors that cause the significant difference in decay rates between strain-rate and rotation-rate.
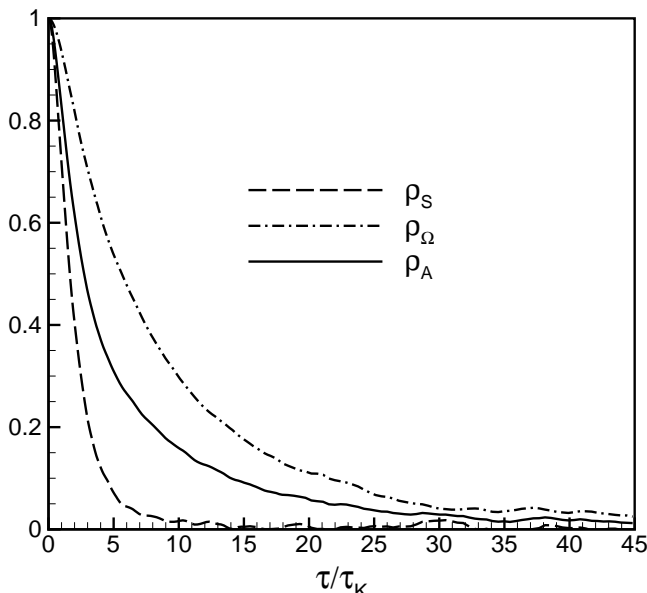

Figure 7.   Lagrangian auto-correlation function of full tensor **A** (solid line), its symmetry part (dash dot line), antisymmetric part (dash line)

One possible factor for the slow decay of rotation-rate could be due to contributions from fluid particles that occur around and near small-scale vortical structures (worms). These structures are known to be relatively long-lived. Inside such structures, the vorticity would be relatively constant, pointing along the axis of the "worm" and thus the rotation tensor would be time persistent in magnitude and direction. The idea then is to recompute the autocorrelation function by systematically including or excluding rotation-dominated flow regions. There are many ways to accomplish this and we have experimented with several. In the end, results pertaining to using the second invariant (Q-criterion) are qualitatively quite similar to those of the other criteria, so those based on the Q-criterion are presented here.

The second invariant of **A** is defined as $Q = -\frac{1}{2}Tr(\mathbf{AA}) = -\frac{1}{2}A_{ij}A_{ji} = \frac{1}{2}(\Omega_{ij}\Omega_{ij} - S_{ij}S_{ij})$ for an incompressible flow ($A_{ii} = 0$). It is often used to identify vortices as flow regions with positive Q, i. e. $Q > 0$ [38]. We undertake analysis using conditional averaging based on the Q-criterion at the initial time of the correlation function ($\tau = 0$), attempting to include ($Q(t_0) > 0$) or exclude ($Q(t_0) < 0$) initial points that are more or less likely to be part of "worms" (elongated rotation-dominated coherent structures). The conditional auto-correlations for rotation-rate **Ω** are thus computed according to

$$\rho_{\mathbf{\Omega}}^+(\tau) \equiv \frac{\langle\Omega_{ij}(t_0)\Omega_{ij}(t_0 + \tau)|Q(t_0) > 0\rangle}{\sqrt{\langle(\Omega_{mn}(t_0))^2|Q(t_0) > 0\rangle\langle(\Omega_{pq}(t_0 + \tau))^2|Q(t_0) > 0\rangle}}, \tag{8}$$

and similarly $\rho_{\mathbf{\Omega}}^-(\tau)$ for $Q(t_0) < 0$.
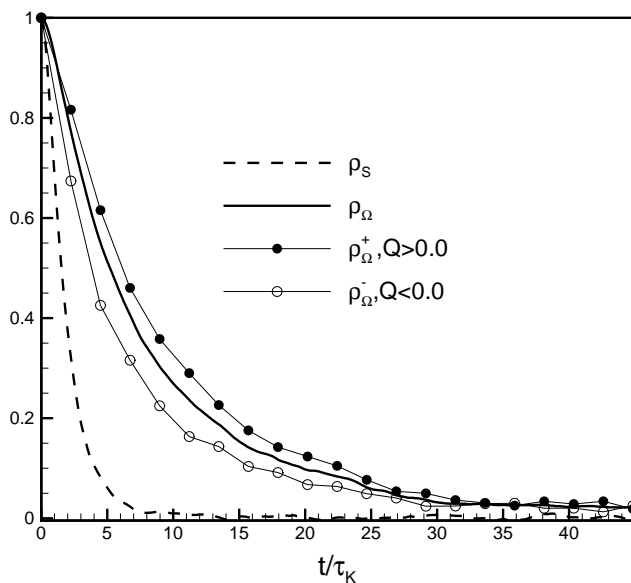
Figure 8.   Conditional auto-correlations of rotation-rate tensor with different $Q(0) > 0$ and $Q(0) < 0$. Lines with filled symbols correspond to $Q(0) > 0$ (dominated by rotation, plausibly more associated to "worms") and lines with empty symbols correspond to $Q(0) < 0$ (exclusion of "worms" by focussing on strain-domination). Solid and dashed lines are non-conditional auto-correlations of the rotation-rate tensor and strain-rate tensor respectively.

In Fig. 8, the line with filled symbols is for positive Q conditional averaging, designed to focus mostly on worms. It should be remarked that flow visualizations have shown that $Q > 0$ isolates quite successfully regions that visually correspond to elongated vortices in turbulence. Higher thresholds can also be used and the trends are qualitatively quite similar to those observed. Clearly Fig. 8 shows that the correlation decay is even slower if one focuses only on the rotation dominated regions. But the difference with the unconditional results is not that particularly large. When conditioning on $Q < 0$, i.e. excluding entirely the rotation dominated regions, the decay is slightly faster than the unconditional results. However, the decay is still considerably slower than the decay of **S**. This demonstrates that the coherent structures ("worms") play a role in the slower decay rate of autocorrelation, but perhaps not a dominant role, and certainly not the only one.

Another possible cause for the rapid decay of strain-rate is the distinct role of the pressure Hessian. The coupled equations (9) and (10) for Lagrangian evolutions of strain- and rotation-rate tensors can be easily derived from the evolution of velocity tensor, Eq. (6) [39, 40].

$$\frac{DS_{ij}}{Dt} = \Omega_{jk}\Omega_{ik} - S_{jk}S_{ik} - P_{ij} + \nu\nabla^2 S_{ij}, \tag{9}$$

$$\frac{D\Omega_{ij}}{Dt} = \Omega_{jk}S_{ik} - S_{jk}\Omega_{ik} + \nu\nabla^2\Omega_{ij}. \tag{10}$$

In the equations, the symmetric pressure-Hessian appears only in the evolution of strain-rate, Eq. (9), but not in the rotation-rate equation (10), implying a direct effect of pressure on strain-rate but only an indirect effect on rotation-rate (through the vortex stretching and tilting by the strain-rate).

We use two ways to examine how pressure affects the dynamics of strain- and rotation-rate. First, we examine the correlation functions of terms in the right-

hand side of Eq. (9) with the rate-of-change of **S** respectively. Specifically, we look at the deviatoric parts of the pressure Hessian, $P_{ij}^d = -[P_{ij} - P_{kk}\delta_{ij}/3]$, and mutual interaction term, $M_{ij}^d = \Omega_{jk}\Omega_{ik} - S_{jk}S_{ik} - 1/3(\Omega_{mk}\Omega_{mk} - S_{np}S_{np})\delta_{ij}$. The correlation coefficient of the rate-of-change of **S**, $\mathbf{a_S} = D\mathbf{S}/Dt$, with these terms is defined as

$$\rho_{a_\mathbf{S}\mathbf{C}} = \frac{\langle a_{S_{ij}} C_{ij}\rangle}{\sqrt{\langle (a_{S_{mn}})^2\rangle\langle(C_{pq})^2\rangle}} \tag{11}$$

where **C** can be $\mathbf{P}^d$ or $\mathbf{M}^d$. We find that the correlation of $\mathbf{P}^d$ with $a_\mathbf{S}$ ($\rho_{a_\mathbf{S}\mathbf{P}} \sim 0.75$) is much larger than that of $\mathbf{M}^d$ with $a_\mathbf{S}$ ($\rho_{a_\mathbf{S}\mathbf{M}} \sim 0.17$) implying that pressure Hessian (its deviatoric part) has a more dominant effect on the dynamics of strain-rate compared to the "velocity gradient self-interaction part". The pressure Hessian depends upon nonlocal flow processes that at any given position, introduce additional randomness. Thus, the fact that the temporal decorrelation of strain-rate is much faster than that of rotation is to be expected if in its evolution equation the effects of pressure Hessian dominate rather than the self-stretching terms.

In order to explore the effects of the various terms further, we investigate the dynamics of S_{ij} and Ω_{ij} systematically by integrating their evolution equations (9) and (10), first including only the inviscid self-stretching terms without pressure or viscosity, then including the pressure Hessian term, and finally, comparing the results to full DNS which also includes the viscous terms. Numerical time integration of the ODEs in Eqs. (9) and (10) is performed using a 4th-order Runge-Kutta algorithm in the cases with only self-stretching, and also with the pressure Hessian available in the JHU database. The values of S_{ij}, Ω_{ij}, and P_{ij} at the start time of the integration are retrieved along the trajectories. Over 10,000 fluid particle locations are tracked using the *GetPosition* function during the time evolution in order to obtain the instantaneous pressure Hessian components from the turbulence database. We compare the time correlations computed from the solutions of Eqs. (9) and (10) with the DNS data analysis. In Fig. 9, solid symbols are for rotation rate while empty symbols are for strain-rate. Three types of results are shown: circles (DNS analysis, i.e. including self-stretching inviscid terms, pressure Hessian, as well as viscous terms), squares (dynamics with self-stretching inviscid terms and pressure Hessian but without viscous terms), and triangles (dynamics with self-stretching inviscid terms but without pressure Hessian and no viscous terms).

As can be seen Fig. 9, for the decay of the correlation function for the strain-rate, the pressure Hessian plays a dominant role in accelerating the decay rate away from the longer time-scales it would have if only the inertial self-stretching terms were retained (triangles). Interestingly, if the pressure Hessian term is included but not the viscous term (squares), the decorrelation is even slightly faster than if the viscosity is included as in the DNS. The difference is not major and we do not have any clear explanation why viscous forces would, in this case, slightly increase the memory of the strain-rate tensor evolution. Conversely, for the rotation memory, inclusion of pressure Hessian changes the decay of correlation very little. The change observed occurs because the strain-rate becomes more rapidly decorrelated with pressure effects, and this modulates the vortex stretching and tilting by the strain-rate tensor. Inclusion of viscous effects reduces the correlation by a small, further amount.
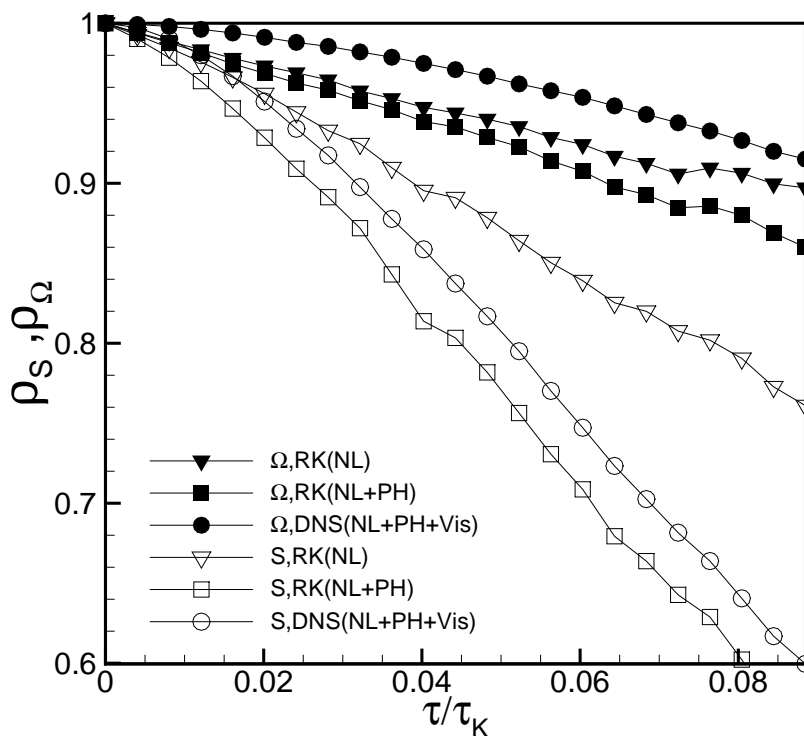
Figure 9.   Lagrangian autocorrelation functions of strain- and rotation-rate tensors obtained through particle tracking in the DNS data (circles), and from a Runge-Kuta integration of strain and rotation tensors according to Eqs. (9) and (10), including pressure Hessian term but without viscous terms (squares) and without pressure Hessian term nor viscous terms (triangles). In all cases, time-integration is done following the same particle trajectories tracked in the database using the *GetPosition* function. NL: Nonlinear term; PH: Pressure Hessian term; Vis: Viscous term.

5.   Testing the Recent Fluid Deformation Approximation to model pressure Hessian

In this subsection, we examine a recently proposed model for the anisotropic pressure Hessian term in Eq. 6 that can be used in stochastic Lagrangian models for the velocity gradient tensor. As background about the model, we recall that assuming the pressure Hessian is isotropic (i.e.neglecting $\partial_{ij}^2 p - \partial_{kk}^2 p\ \delta_{ij}/3$) and neglecting the viscous term in Eq. 6 leads to a closed formulation for $\mathbf{A}$, the so-called Restricted-Euler (RE) equation. The RE system is a set of 9 (8 independent) ordinary differential equations for A_{ij} that has analytical solutions [41]. Remarkably this simple system is already sufficient to explain a number of non-trivial geometrical trends found in real turbulence [41, 42]. Nevertheless, the RE system leads to nonphysical finite-time singularities because the self-stretching is not constrained by any energy exchange or loss mechanism in the system. In the past two decades, modeling efforts have aimed at regularizing the RE system to avoid the nonphysical singularity (see [13] for a review). One of the efforts is the recent fluid deformation approximation (RFDA) as proposed in Ref. [43]. The starting point of RFDA is an Eulerian-Lagrangian change of variables

$$\frac{\partial^2 p(\mathbf{x}, t)}{\partial x_i \partial x_j} \approx \frac{\partial x_{p,m}}{\partial x_i} \frac{\partial x_{p,n}}{\partial x_j} \frac{\partial^2 p(\mathbf{x}, t)}{\partial x_{p,m} \partial x_{p,n}} \tag{12}$$

where spatial gradients of $D_{ij} = \frac{\partial x_i}{\partial x_{p,j}}$ are neglected. As mentioned above, $\mathbf{x}$ and $\mathbf{x}_p$ denote the Eulerian space location and Lagrangian particle location respectively.

The Lagrangian pressure Hessian, $\frac{\partial^2 p(\mathbf{x},t)}{\partial x_{p,m} \partial x_{p,n}}$, is modeled as an isotropic tensor based on the assumption that as time progresses, one loses memory about the relative orientations of the initial locations $\mathbf{x}_p$ as far as the present value of pressure is concerned. By introducing the Cauchy-Green tensor $\mathbf{C}$, $C_{ij} \equiv \frac{\partial x_i}{\partial x_{p,m}} \frac{\partial x_j}{\partial x_{p,n}}$, and using the Poisson equation as a constraint, the pressure Hessian becomes

$$\frac{\partial^2 p(\mathbf{x},t)}{\partial x_i \partial x_j} = -\frac{Tr(\mathbf{A}^2)}{Tr(\mathbf{C}^{-1})} C_{ij}^{-1} \tag{13}$$

Based on the idea that any causal relationship between initial and present orientations will be lost after a characteristic Lagrangian correlation time scale of the tensor $\mathbf{A}$, the Cauchy-Green tensor $\mathbf{C}$ in Eq. (13) is further replaced by a new tensor called the "recent Cauchy-Green tensor" $\mathbf{C}_{\tau_K}$ that can be expressed in terms of simple matrix exponentials $\mathbf{C}_{\tau_K} = e^{\tau_K \mathbf{A}} e^{\tau_K \mathbf{A}^T}$.

The trace of the pressure Hessian requires no modeling, since it is equal to the trace of $-\mathbf{A}^2$, by construction in the model, and by the incompressibility condition in real turbulence. Hence, we only examine the deviatoric part of this tensor. We compare the temporal auto-correlations of the deviatoric pressure Hessian by using the tensor-based correlation function as defined in Eq. 7. Particle tracks are evaluated using *GetPosition* function using over 10,000 particles. The model uses matrix exponential evaluations [44]. The results are shown in Fig. 10. The dashed line is from the RFDA-based model, and the solid line from the DNS. It is seen that the auto correlations computed from the model decay more slowly then the DNS. The deviatoric pressure Hessian in DNS looses most of its memory at $\tau \approx 1.5\tau_K$ whereas the model term maintains some memory up to $\tau \approx 4\tau_K$. While the model has shown promise in predicting many features of the velocity gradient tensor in turbulence [43, 45], challenges remain in applications at high Reynolds numbers. The present observations of differing correlation times may point to possible improvements in the model.

Next, we test how the modeled pressure Hessian captures features of individual realizations and time series of the two most relevant invariants of the velocity gradient $\mathbf{A}$, the Q and R invariants. Q has already been defined as $Q = -\frac{1}{2} A_{ij} A_{ij}$, while R is given by $R = -\frac{1}{3} A_{ij} A_{jk} A_{ki}$. Physically these invariants are interpreted as quantifying the competition of enstrophy vs. dissipation, and of enstrophy production vs. dissipation production, respectively. The dynamics and statistics of these two variables has attracted much interest. Their evolution equations are derived by forming appropriate products with Eq. (6) and taking the trace [41]:

$$\frac{dQ}{dt} = -3R - A_{ik} P_{ki} - A_{ik} V_{ki} \tag{14}$$

$$\frac{dR}{dt} = \frac{2Q^2}{3} - A_{ij} A_{jk} P_{ki} - A_{ij} A_{jk} V_{ki} \tag{15}$$

where $V_{ij} \equiv \nu \partial^2 A / \partial x_i \partial x_j$.

We track a single particle and record a time-series of relevant terms, across the entire time range available in the database. Figure 11 shows the various terms from the DNS. The dashed line is the rate of change of Q and R as evaluated from their database values along the trajectory, the circles are the restricted Euler self-stretching term, the solid line comes from the pressure Hessian, and the triangles is the viscous term. The viscous term is evaluated based on taking the difference
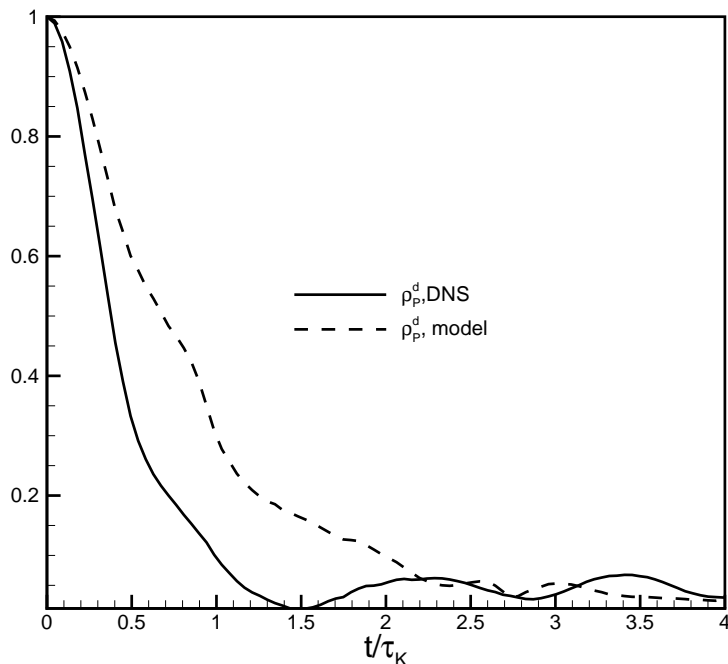
Figure 10. Lagrangian auto-correlation function of deviatoric pressure Hessian from DNS (solid line) and modeling (dash line).

of the other terms on the right-hand-side to the temporal rates of change of Q and R.

As can be seen, the dynamics are highly intermittent, with a sudden, rapid burst of activity near $t/\tau_K \sim 31$ for this particular fluid particle's history. It is observed that the pressure Hessian is the major contribution to the sum and it mainly opposes the self-stretching, whereas the viscous term is small and contributes only marginally. Very interestingly, close examination shows that the pressure Hessian has a "phase delay" that follows the rapid changes in the velocity gradient invariants. The delay appears to be of the order of $\sim \frac{1}{2}\tau_K$.

Next, the ability of the RFDA-based model of Ref. [43] for pressure Hessian to predict the effects on the invariants is shown by providing an enlarged view of the time-series, in the vicinity of where the burst of activity is observed. In Fig. 12 we compare the DNS pressure Hessian term with that predicted by the RFDA model, where the pressure Hessian contracted with A_{ij} and $A_{ik}A_{kj}$ is obtained from the RFDA-based model. Qualitatively there is a general agreement of the occurrences of large peaks, and their signs. However, the model amplitudes appear to be somewhat too large, and the model does not predict some of the smaller-amplitude fluctuations. It is also quite obvious that the model "predates" the real pressure Hessian by about $\sim \frac{1}{2}\tau_K$, which is not surprising since it is based on the local velocity gradient tensor through the matrix exponential closure. Finally, the previous observations can be made more quantitative by computing the two-time cross-correlation function between the real and modeled pressure Hessian tensors. We use an expression similar to Eq. 7 written as a cross-correlation between two different tensor time signals. In particular, in Eq. 7 we take $C_{ij}(t_0)$ to be the modeled pressure Hessian, and $C_{ij}(t_0 + \tau)$ to be the real Hessian tensor. Figure 13 shows the resulting cross-correlation function. It confirms the prior observations: there is a peak correlation after a time-delay of about $\sim \frac{1}{2}\tau_K$, so that the model predates the real pressure Hessian signal in time. The correlation peak of around 40% is quite substantial, given the many assumptions made in deriving the model.
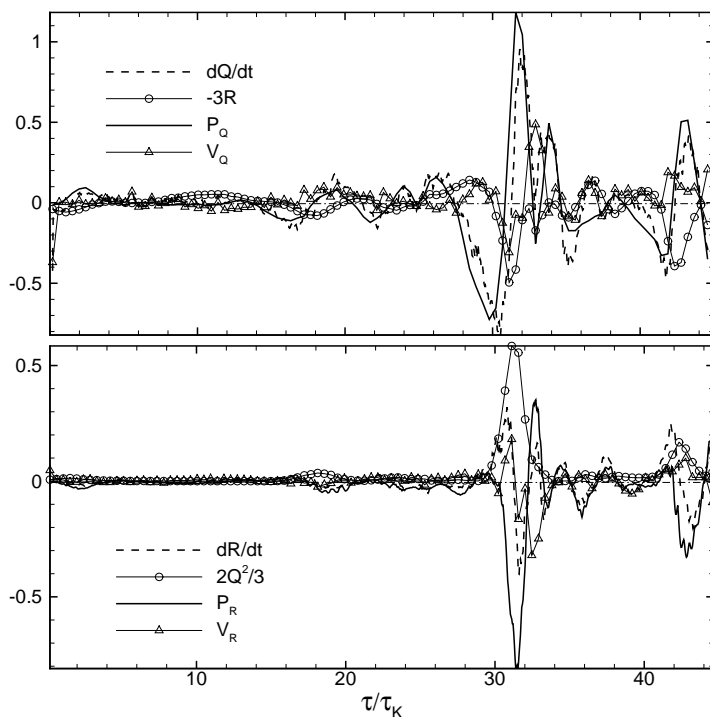
Figure 11. Sample time series for the various terms in the evolution of Q (top) and R(bottom) for some fluid particle along its trajectory during the entire time duration of the database (approximately one large-eddy turnover time).

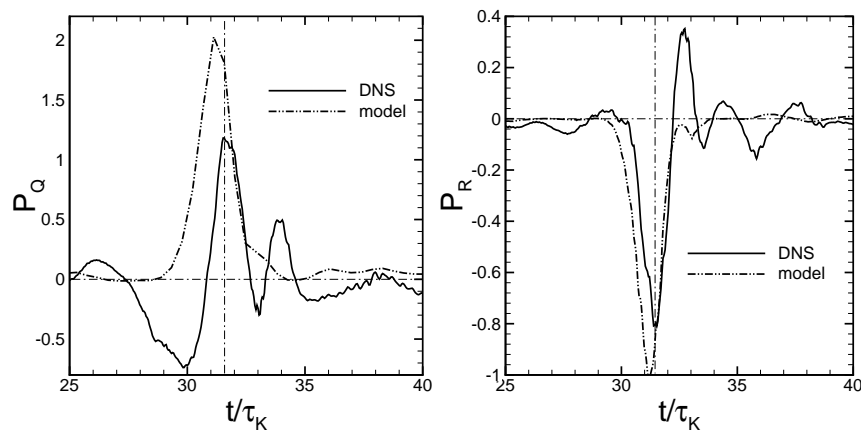Such observations will be useful in motivating further improvements to the model.

Figure 12. Contributions of pressure Hessian to the dynamics of Q (left) and R (right). DNS: solid line; model: dash line.

6.   Summary and discussion

This paper describes algorithms and implementation details of updates to the JHU turbulence public database system, made after the first publication [2] describing the original system. The updates include new *GetFunction*s, namely *GetPosition* to track number of fluid particles moving along with the simulated flow and is useful in Lagrangian studies of turbulence. Also, the *GetForce* function is developed in
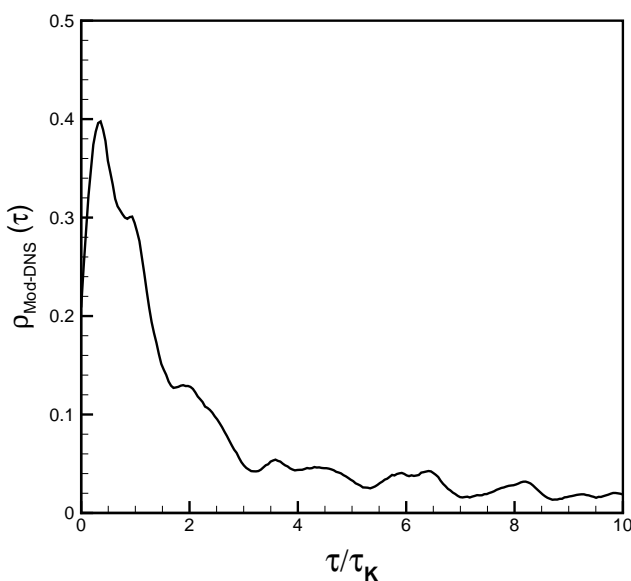
Figure 13.   Two-time cross-correlation function between real and modeled pressure Hessian (its deviatoric part).

order to query the forcing term that was used in the DNS during the simulation (see Appendix A).

Table 2 lists the complete *getFunction*s available to use.

Function name	Spatial diff.	Spatial int.	Temporal int.	Outputs
GetVelocity	–	NoInt, Lag4,6,8	NoInt, PCHIP	u_i
GetVelocityAndPressure	–	NoInt, Lag4,6,8	NoInt, PCHIP	u_i, p
GetVelocityGradient	FD4,6,8	NoInt, Lag4,6,8	NoInt, PCHIP	$\frac{\partial u_i}{\partial x_j}$
GetPressureGradient	FD4,6,8	NoInt, Lag4,6,8	NoInt, PCHIP	$\frac{\partial p}{\partial x_i}$
GetVelocityHessian	FD4,6,8	NoInt, Lag4,6,8	NoInt, PCHIP	$\frac{\partial^2 u_k}{\partial x_i \partial x_j}$
GetPressureHessian	FD4,6,8	NoInt, Lag4,6,8	NoInt, PCHIP	$\frac{\partial^2 p}{\partial x_i \partial x_j}$
GetVelocityLaplacian	FD4,6,8	NoInt, Lag4,6,8	NoInt, PCHIP	$\frac{\partial^2 u_i}{\partial x_j \partial x_j}$
GetForce	–	NoInt, Lag4,6,8	NoInt, PCHIP	f_i
GetPosition	–	Lag4,6,8	PCHIP	$x_i(t_{ET})$

Table 2.   List of *GetFunctions* for queries to the JHU turbulence public database. The entries mean: diff – differentiation (FD: Centered finite difference, options for 4th-, 6th-, and 8th-order accuracies); int – interpolation type (NoInt: no interpolation; Lag: Lagrangian polynomial interpolation, options for 4th-, 6th-, and 8th-order accuracies); PCHIP: Piecewise cubic Hermite interpolation.

Other recent upgrades also include improved interpolation schemes (Appendix B) and a new library for Matlab access (Appendix C).

The new *GetPosition* function was applied to measure various Lagrangian statistical features of turbulence. In terms of Lagrangian structure functions, we document good agreement with a variety of previously published results, both numerical and experimental. New results are obtained in characterizing the precise effects of pressure Hessian and viscous terms in the Lagrangian evolution of the strain-rate and rotation tensors. The faster decay of autocorrelation for the strain-rate tensor is confirmed to be, clearly, related to the pressure Hessian effects. They tend to be more "stochastic" than the self-stretching terms. The viscous terms were seen to slightly enhance the memory for the strain-rate, while decreasing memory for the rotation rate (or vorticity).

The new tool was also used to examine the time evolution of the pressure Hessian and to compare it with a recent model based on the local velocity gradient tensor. The comparisons were made using the Lagrangian autocorrelation function and its rate of decay, comparing the DNS with the model. It was found that the model decays more slowly, showing that the true pressure Hessian has dynamics that are shorter-lived than the velocity gradients upon which the model is based. Some representative observations about the model were also made on hand of individual time traces along Lagrangian trajectories, comparing terms in the equations of invariants Q and R. It is observed that the pressure Hessian 'lags' strong excursions in velocity gradients, consistent with a "restitution mechanism" that needs some time to build up the required response.

Further developments of the public database system are being sought, including additional datasets such as magneto-hydrodynamic turbulence, and turbulent channel flow.

Appendix A. *GetForce* **function**

Information about the forcing term $f_i(x, y, z, t)$ (force per unit mass, $i = x, y, z$) applied during the DNS has been stored in the database and can be retrieved using the function *GetForce*.

During DNS, an effective forcing is applied in Fourier space by rescaling low-k Fourier modes (with magnitudes $0.5 \leq k \leq 2.5$, $k = \sqrt{k_x^2 + k_y^2 + k_z^2}$) to maintain their kinetic energy to prescribed values consistent with a $-5/3$ spectrum. The forcing region is divided into two shells, $0.5 \leq k \leq 1.5$ and $1.5 < k \leq 2.5$. The spectrum is held fixed at a value of 0.3 in shell $0.5 \leq k \leq 1.5$, and at a value equal to 0.13 in shell $1.5 < k \leq 2.5$ shell (these values are obtained empirically so that the simulated spectrum is close to a $k^{-5/3}$ trend at low k).

In order to represent the rescaling in terms of a forcing term, we express the time-advancement in terms of a first-order time-advancement and write the discretized Navier-Stokes equation (NSE) in Fourier space as follows

$$\hat{u}_i^{n+1}(k_x, k_y, k_z) = \hat{u}_i^{n+}(k_x, k_y, k_z) + \hat{f}_i(k_x, k_y, k_z)dt \tag{A1}$$

in which $\hat{u}_i^{n+} = \hat{u}_i^n + (\cdots)dt$ with $(\cdots)$ for terms on the right-hand side of the Navier-Stokes equations, but excluding the forcing term. Also, dt is the time-step of the DNS.

In the DNS, the rescaling induces a difference between $\hat{u}_i^{n+}$ and $\hat{u}_i^n$ in the wave-number range $0.5 \leq k \leq 2.5$ that is equivalent to a force-term defined in the two shells as follows

$$\hat{f}_i^n(k_x, k_y, k_z) = \frac{1}{dt}\left(\frac{0.55}{\sqrt{\sum_{0.5 \leq k \leq 1.5}[(\hat{u}_x^{n+})^2 + (\hat{u}_y^{n+})^2 + (\hat{u}_z^{n+})^2)]/2}} - 1\right)\hat{u}_i^{n+}(k_x, k_y, k_z)$$

$$\tag{A2}$$

for shell $0.5 \leq k \leq 1.5$ and

$$\hat{f}_i^n(k_x, k_y, k_z) = \frac{1}{dt} \left(\frac{0.36}{\sqrt{\sum_{1.5 \leq k \leq 2.5}[(\hat{u}_x^{n+})^2 + (\hat{u}_y^{n+})^2 + (\hat{u}_z^{n+})^2)]/2}} - 1 \right) \hat{u}_i^{n+}(k_x, k_y, k_z)$$
(A3)

for shell $1.5 < k \leq 2.5$, where $\hat{u}_x$, $\hat{u}_y$, $\hat{u}_z$ denote the three velocity components in Fourier space and $k = \sqrt{k_x^2 + k_y^2 + k_z^2}$ is the magnitude of wavenumber vector **k**. In this way, the energy in these shells $E(k=1) = \sum_{0.5 \leq k \leq 1.5}(\hat{u}_x^2 + \hat{u}_y^2 + \hat{u}_z^2)/2$ and $E(k=2) = \sum_{1.5 < k \leq 2.5}(\hat{u}_x^2 + \hat{u}_y^2 + \hat{u}_z^2)/2$ is maintained at 0.3 and 0.13.

There exist in total 80 discrete wave-number modes in these two shells. There are 20 modes for $k_x = 0$, 30 modes for $k_x > 0$, and another 30 modes for $k_x < 0$. In the database, the complex Fourier coefficients $\hat{f}_x$, $\hat{f}_y$, $\hat{f}_z$ corresponding to $k_x \geq 0$ (50 modes) are stored, the remaining 30 modes ($k_x < 0$) are the conjugates of the modes $k_x > 0$.

Using the *GetForce* function, force values at any prescribed position (x, y, z) are evaluated in the database from the forcing's Fourier coefficients using direct summation of the Fourier series, according to

$$f_i(x, y, z, t_n) = \sum_{k_x, k_y, k_z} e^{i(k_x x + k_y y + k_z z)} \hat{f}_i^n(k_x, k_y, k_z)$$
(A4)

where i can be x, y, and z. Values of $f_i(x, y, z, t)$ at arbitrary times t can be obtained by specifying PCHIP temporal interpolation.

In order to document the use of this function, we examine various terms in the Navier-Stokes equations

$$\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \nabla^2 \mathbf{u} + \mathbf{f}$$
(A5)

that were solved during DNS (as explained above, the forcing term is implicitly included in the spectral rescaling at every time-step). We evaluate the local square error defined according to

$$\sigma_{dif}^2 = \langle [\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u} - (-\nabla p + \nabla^2 \mathbf{u} + \beta \mathbf{f})]^2 \rangle.$$
(A6)

The goal is to compare the case where we include ($\beta = 1$) and do not include ($\beta = 0$) the forcing term. We would expect that including the forcing term should reduce the error. If we evaluate the velocity gradients occurring in the nonlinear term, the pressure gradient, and the viscous Laplacian using pseudo-spectral differentiation, and use the same time-differentiation as used in the DNS, the error should be exactly zero (to machine accuracy) at every point in the domain. However, if we use the spatial finite differencing available in the *getFunction*s, and the first-order time derivative, some error is expected. Instead, if we box-filter each of the terms in boxes of size $\Re$, with increasing $\Re$ the error would be expected to become smaller. Especially the difference between including and not including the forcing term (which by construction only affects the very largest scales of the flow), is expected to become larger as $\Re$ grows.

Thus, we also define the error associated with the coarse-grained terms, according to

$$\sigma_{dif,\Re}^2 = \langle \left([\partial_t \mathbf{u} + \mathbf{u} \cdot \nabla \mathbf{u}]_\Re - \left[(-\nabla p + \nabla^2 \mathbf{u} + \beta \mathbf{f}]_\Re \right) \right)^2 \rangle. \tag{A7}$$

The square brackets $[...]_\Re$ denote box filtering in a cube of size $\Re$. In Figure A1 we show dependence of the rms error $\sigma_{dif,\Re}$ as function of $\Re$.

The computation of σ_{dif} follows three steps: First, randomly generate N cubes with size $\Re$ in the whole domain; second, collect all the terms in Eq. (A5) by calling *getVelocity, getVelocityGradient* for the left-hand side and *getPressureGradient, getVelocityLaplacian,* and *getForce* for the right-hand side for every point in each cube and compute the mean of each term; third, evaluate the square error. The filter size $\Re$ varies from 0.006 to 0.3, corresponding 1 to 49 grid-points. In the figure, it is seen that when the filter size is small, say $\Re < 0.02$, forcing seems not to play a role because the numerical errors introduced from differentiation and time/spatial interpolations dominate and suppress any effects of the forcing term. As the filter size increases the numerical errors fade such that the forcing term becomes more important. When the filter size is large enough, e.g. $\Re > 0.2$, the difference between left- and right- hand side of Eq. (A5) vanish when forcing is included, while $\sigma_{rms,\Re}$ without forcing remains quite large making the effects of the forcing term apparent in closing the balance in the momentum equation.
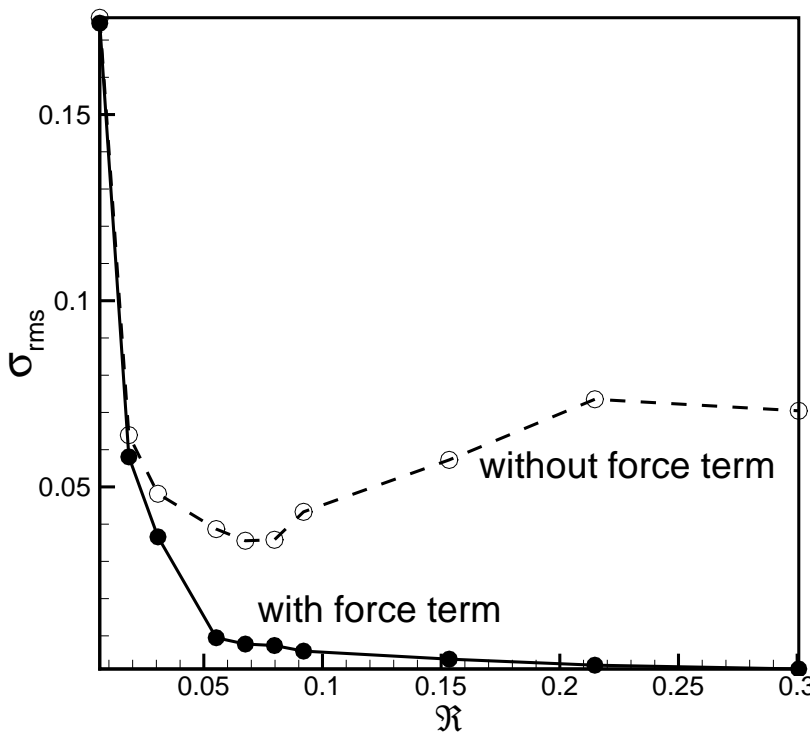

Figure A1.   Magnitude of error between left- and right-hand side of Eq. (A5) and dependence on the box filtering size $\Re$, including the force term (closed circles) and not including the force term (empty circles).

Appendix B. Partial sums evaluation for interpolations

We have implemented a new method for the evaluation of the spatial interpolation used in the database routines, which we summarize below. The method is described

in more detail in [46]. It is also applicable to the temporal interpolation and differentiation evaluations, but we are still in the process of implementing it for those computations.

The database routines perform Lagrange polynomial interpolation of order specified by the user. For N^{th} order Lagrange polynomial interpolation of a point $\mathbf{p}'$ in 3-d space we have:

$$f(\mathbf{p}') = \sum_{k=1}^{N} l_z^{q-\frac{N}{2}+k}(z') \sum_{j=1}^{N} l_y^{p-\frac{N}{2}+j}(y') \sum_{i=1}^{N} l_x^{n-\frac{N}{2}+i}(x')$$

$$\cdot f(x_{n-\frac{N}{2}+i}, y_{p-\frac{N}{2}+j}, z_{q-\frac{N}{2}+k}) \ . \tag{B1}$$

In the above $\mathbf{p}' = (x', y', z')$ is the target location and the data stored in the database at location (x_i, y_j, z_k) is given by $f(x_i, y_j, z_k)$. Since data in the database are stored at the nodes of a discrete grid, the grid location (x_n, y_p, z_q) is computed as $n = int(\frac{x'}{\Delta x} + \frac{1}{2})$, $p = int(\frac{y'}{\Delta y} + \frac{1}{2})$, $q = int(\frac{z'}{\Delta z} + \frac{1}{2})$, where Δx, Δy, Δz are the widths of the grid in the x, y and z dimensions. The Lagrange coefficients l in Eq. (B1) are as follows:

$$l_\theta^i(\theta') = \frac{\displaystyle\prod_{j=\alpha-\frac{N}{2}+1, j\neq i}^{\alpha+\frac{N}{2}} (\theta' - \theta_j)}{\displaystyle\prod_{j=\alpha-\frac{N}{2}+1, j\neq i}^{\alpha+\frac{N}{2}} (\theta_i - \theta_j)} \ , \tag{B2}$$

where θ can be x, y or z and α can be n, p or q, respectively.

The evaluation of the Lagrange polynomial interpolation requires data from a cube of width N around the target location. In the current version of the database edge overlap ensures that all of the necessary data are contained within a single database atom, and hence a single database I/O is needed to perform the computation. However, in general the data may be spread across multiple such database atoms or even across multiple database servers. In order to ensure the efficient processing of large batch queries submitted by our users we evaluate the interpolation by means of partial sums.

Lagrange polynomial interpolation as well as any other linear computation can be executed in parts by maintaining and updating a partial sum of the final result. We make use of this observation to evaluate multiple target positions at the same time and by means of a single, sequential pass over the data. We process all target positions in a user's batch and determine the entire set of database atoms that need to be read from the database in order to perform the interpolation of each target. For each database atom read from the database we increment the partial sums of all target positions, whose interpolation kernel intersects the database atom. Once all such atoms are processed the interpolation of each target position has been evaluated.

We make use of the efficient procedures of Purser and Leslie [47] in the computation of the Lagrange coefficients. They present efficient ways to organize the coding, eliminating redundant multiplications and making use of the fact that the values in the denominator of Eq. (B2) are constant to reduce the time complexity of the computation of the coefficients to $O(N)$ from $O(N^3)$.

Appendix C. Matlab interface

The Matlab interface allows clients to interact with the turbulence database directly
from a Matlab session. This interface is based on Matlab web service functions
which communicate with the database directly using the Simple Object Access
Protocol (SOAP). All communication with the JHU Turbulence Database Clus-
ter is controlled through the TurbulenceService Matlab class. This class creates
SOAP messages, queries the database, and parses the database response. For each
database function a wrapper function has been created to perform the data trans-
lation and retrieval. One major advantage of the Matlab interface to that of its
C and Fortran counterparts is the readily available functions and toolboxes that
Matlab provides. With the Matlab interface, clients can retrieve sections of spa-
tiotemporal data from the database, view the data with Matlab's plotting tools or
perform secondary calculations on the data, all from the same Matlab session.

A standard distribution of Matlab contains a set of functions for creating (*create-
SoapMessage*), sending (*callSoapService*) and parsing (*parseSoapResponse*) SOAP
messages. These routines use a W3C compliant Document Object Model (DOM)
approach for constructing and parsing the Extensible Markup Language (XML)
formatted SOAP message. The DOM provides a generic mechanism to create XML
documents. However, while being robust and dynamic, the DOM approach holds
the disadvantage of being computationally inefficient for large XML documents
– this inefficiency becomes a limiting factor for large database queries. To avoid
this critical problem, we have developed faster replacement functions to create and
send the SOAP message, and to parse the SOAP response. Therefore, by perform-
ing low-level string operations instead of the employing the DOM, we can rapidly
build and parse extensive XML documents leading to a $100x$ speedup over the
original DOM approach. Due to this increase in efficiency, the Matlab interface
possess similar performance characteristics as those of the C and Fortran database
interfaces.

The basis for the Matlab database functions are created by using the *create-
ClassFromWsdl* utility. This utility generates the TurbulenceService Matlab class
from the Web Service Definition Language (WSDL) functions of the database web
service. These generated files are modified to incorporate the newly developed
faster Matlab SOAP routines. The purpose of the TurbulenceService class is to
accommodate a request to the database by taking data from Matlab, generating
an appropriate SOAP message, sending the message to the database and finally
retrieving and parsing the database response. While providing a direct mechanism
for interacting with the database, the TurbulenceService class returns data pack-
aged in a Matlab structure array which may not be necessarily intuitive to most
Matlab users. We have, therefore, created wrapper functions which translate the
response structure into directly accessible Matlab vectors.

The following code snippets illustrate the complete mechanism, starting from
user-generated request data and ending with a parsed database response, stored
in *response*. From a Matlab script, request data will be provided to the *getVelocity*
TurbulenceService wrapper function as demonstrated in Listing 1. This wrapper
function calls the TurbulenceService *TS_getVelocity* function (see Listing 2), and
translates its structure into a vector of velocity components. The *TS_getVelocity*
function illustrated in Listing 3 assembles the data in a Matlab structure, cre-
ates the SOAP message, sends the SOAP message and parses the SOAP response.
(A similar TurbulenceService is implemented for the *getPosition* function, as illus-
trated in Listing 4).

For illustration of *getVelocity*, in Figure C1(a) is a velocity contour plot of sample

Listing 1   Example call to getVelocity from Matlab interface

```
% Set client authentication key
authkey = '...';
% Set target database
dataset = 'isotropic1024coarse';
% Set temporal interpolation scheme
temporal = 'PCHIP';
% Set spatial interpolation scheme
spatial = 'Lag6';

% Create a set of (x,y,z)-coordinates to query at a randomly
% chosen time step
points(1:3,:) = ...;
time = 0.002 * randi(1024, 1);

% Call TurbulenceService wrapper to perform getVelocity request at
% specified points
response = getVelocity(authkey, dataset, time, ..., points);
```

data from the turbulence database. The data was retrieved using the *getVelocity* function from the Matlab interface and the contour plot was generated using Matlab standard countour plotting tools. In Figure C1(b) a visualization of 'worms' i shown in a small subcube of the data at $t = 0$, using iso-Q surfaces generated using the Matlab implementation of *getVelocityGradients* to evaluate A_{ij}, computing the invariant $Q = -\frac{1}{2}A_{ij}A_{ji}$ at every point in Matlab, and using Matlab 3D plotting tools.
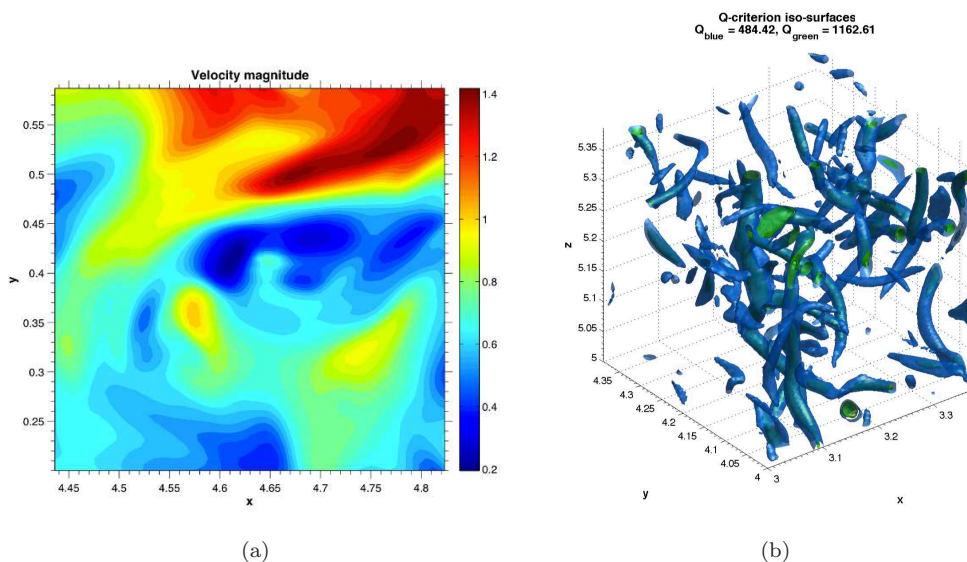

(a)   (b)

Figure C1.   (a) Velocity contour plot generated using the Matlab interface as available for download. (b) Visualization of 'worms' using iso-Q surfaces in a small subset of the data, generated using the Matlab implementation of *getVelocityGradients* to evaluate A_{ij}, computing the invariant $Q = -\frac{1}{2}A_{ij}A_{ji}$ at every point and using Matlab 3D plotting tools.

26   *Huidan Yu,* et al – *PREPRINT*

Listing 2   Sample getVelocity wrapper function

```
function response = getVelocity(authkey, dataset, time, ..., points)

  % Create the TurbulenceService object and call TS_getVelocity
  obj = TurbulenceService;
  responseStruct = TS_getVelocity(obj, authkey, dataset, ..., points);

  % Return a vector of velocity components
  response = getVector(resultStruct.GetVelocityResult.Vector3);

end
```

Listing 3   Sample TS_getVelocity TurbulenceService class function

```
function responseStruct = TS_getVelocity(obj, authkey, dataset, ..., points)

  % Construct a Matlab structure containing the data
  data = struct('points', struct('x', points(1,:), ...), ...);

  % Create the XML document, call the service and parse the response
  soapMessage = createSoapMessage('GetVelocity', data, ...);
  response = callSoapService(URL, soapMessage, ...);
  responseStruct = parseSoapResponse(response);

end
```

Listing 4   Example call to getPosition from Matlab interface

```
% Set client authentication key
authkey = '...';
% Set target database
dataset = 'isotropic1024coarse';
% Set spatial interpolation scheme
spatial = 'Lag6';

% getPosition integration settings
startTime=0.364;
endTime=0.376;
lagDt=0.0004;

% Create a set of (x,y,z)-coordinates
points(1:3,:) = ...;

% Call TurbulenceService wrapper to perform getPosition request at
% specified points between startTime and endTime
response = getPosition(authkey, dataset, startTime, endTime, lagDt,... , points);
```

..

References

[1] http://turbulence.pha.jhu.edu.

[2] Y. Li, E. Perlman, M. Wan, Y. Yang, R. Burns, C. Meneveau, R. Burns, S. Chen, A. Szalay, and G. Eyink, *A public turbulence database cluster and applications to study Lagrangian evolution of velocity increments in turbulence*, J. Turbulence 9 (2008), N31.

[3] P. K. Yeung, S. B. Pope, & B. L. Sawford, *Reynolds number dependence of Lagrangian statistics in large numerical simulations of isotropic turbulence*, J. Turbulence 7 (2006), N58.

[4] S. Hoyas & J. Jimenez, (2006) *Scaling of velocity fluctuations in turbulent channels up to $Re_\tau = 2000$*, Phys. of Fluids 18 (2006), 011702.

[5] J. Schumacher, K. R. Sreenivasan, and V. Yakhot, *Asymptotic Exponents from Low-Reynolds number flows*, New J. Phys., 89 (2007), $1 - 19$.

[6] T. Ishihara, T. Gotoh, & Y. Kaneda, *Study of high Reynolds number isotropic turbulence by direct numerical simulation*,Annu. Rev. Fluid Mech. 41 (2009), $4165 - 80$.

[7] B. Lüthi,M. Holzner, & A. Tsinober, *Expanding the QR space to three dimensions*, J. Fluid Mech. 641 (2009), pp.$497 - 501$.

[8] H. Yu and C. Meneveau, *Lagrangian Refined Kolmogorov Similarity Hypothesis for Gradient Time-evolution in Turbulent Flows*, Phys. Rev. Lett. 104(2010), pp. 084502.

[9] H. Yu and C. Meneveau, *Scaling of conditional Lagrangian time correlation functions of velocity and pressure gradient magnitudes in isotropic turbulence*, Flow,Turbulence and Combustion, 85 (2010), pp. $457 - 472$.

[10] M. Holzner, M. Guala, B. Lüthi, A. Liberzon, N. Nikitin, W. Kinzelbach, and A. Tsinober, *Viscous tilting and production of vorticity in homogeneous turbulence*, Phys. Fluids 22 (2010), pp. 061701-1 $- 4$.

[11] A. G. Gungor & S. Menon, *A new two-scale model for large eddy simulation of wall-bounded flows*, Prog. Aero. Sci., 46 (2010), $28 - 45$.

[12] W. Liu and E. Ribeiro, *Scale and Rotation Invariant Detection of Singular Patterns in Vector Flow Fields*, SSPR & SPR Proceedings of the 2010 joint IAPR international conference on Structural, syntactic, and statistical pattern recognition, LNCS 6218, pp. 522531.

[13] C. Meneveau, *Lagrangian dynamics and models of the velocity gradient tensor in turbulent flows*, Annu. Rev. Fluid Mech. 43 (2011), $219 - 245$.

[14] C. C. Wu and T. Chang, *Rank-Ordered Multifractal Analysis (ROMA) of probability distributions in fluid turbulence*, Nonlin. Processes Geophys., 18 (2011), $261 - 268$.

[15] G. L. Eyink, *Stochastic flux freezing and magnetic dynamo*, Phys. Rev. E, 83, 056405.

[16] F. Toschi *iCFDdatabase2, http://mp0806.cineca.it/icfd.php* (examined December 2011).

[17] G. I. Taylor, *Diffusion by continuous movements*, Proc. London Math. Soc. 20 (1921), $196 - 212$.

[18] L. F. Richardson, *Atmospheric diffusion shown on a distance-neighbor graph*, Proc. Roy. Soc. London, Ser. A 110 (1926), $709 - 737$.

[19] A. N. Kolmogorov, *The local structure of turbulence in incompressible viscous fluid for very large Reynolds numbers*, Dokl. Akad. Nauk SSSR 30 (1941), $301 - 314$; also Proc. R. Soc. A 434 (1991), 9 $- 13$.

[20] B.I. Shraiman, E.D. Siggia, *Scalar turbulence*, Nature 405(2000), pp. $639 - 646$.

[21] B. Sawford, *Turbulent relative dispersion*, Annu. Rev. Fluid Mech. 33 (2001), pp. 289317.

[22] P. K. Yeung, *Lagrangian investigations of turbulence*, Annu. Rev. Fluid Mech. 34 (2002), pp. $115 - 142$.

[23] F. Toschi and E. Bodenschatz, *Lagrangian properties of particles in turbulence*, Annu. Rev. Fluid Mech. 41 (2009), pp. $375 - 404$.

[24] G. Falkovich, K. Gawedzki, and M. Vergassola, *Particles and fields in fluid turbulence*, Rev. Mod. Phys. 73 (2001), pp. $913 - 975$.

[25] M. Holzner, A. Liberzon, N. Nikitin, B. Lüthi, W. Kinzelbach, and A. Tsinober, *A Lagrangian investigation of the small-scale features of turbulent entrainment through particle tracking and direct numerical simulation*, J. Fluid Mech. 598 (2008), pp.$465 - 475$.

[26] S. B. Pope, *Lagrangian PDF methods for turbulent flows*, Annu. Rev. Fluid Mech. 26 (1994), pp. 23 $- 63$.

[27] R. Zimmermann, H.T. Xu, Y. Gasteuil, M, Bourgoin, R. Volk, JF Pinton, E. Bodenschatz,*The Lagrangian exploration module: An apparatus for the study of statistically homogeneous and isotropic turbulence*, Rev Sci. Instru., 81 (2010), 055112.

[28] D. H. Kelley and N. T. Ouellette, *Separating stretching from folding in fluid mixing*, Nature Phys. 7 (2011), $477 - 480$.

[29] Szalay, A., S. P. Z. Kunszt, A. Thakar, J. Gray, D. Slutz, and R. J. Brunner, Designing and mining multi-terabyte Astronomy archives: the Sloan Digital Sky Survey, In Proceedings of the ACM SIGMOD International Conference on Management of Data, 2000.

[30] U. Frisch, *Turbulence: The legacy of A. N. Kolmogorov*, Cambridge University Press, Cambridge, 1995.

[31] L. Biferale, E. Bodenschatz, M. Cencini, A. S. Lanotte, N. T. Ouellette, F. Toschi, and H. Xu, *Lagrangian structure functions in turbulence: A quantitative comparison between experiment and direct numerical simulation*, Phys. Fluids 20 (2008), pp. 065103-1 $- 12$.

[32] L. Biferale, G. Boffetta, A. Celani, A. Lanotte and F. Toschi, *Particle trapping in three-dimensional fully developed turbulence*, Phys. Fluids 17 (2005), 021701.

[33] R. Benzi, L. Biferale, E. Calzavarini; D. Lohse, and F. Toschi, *Velocity-gradient statistics along particle trajectories in turbulent flows: The refined similarity hypothesis in the Lagrangian frame*, Phys. Rev. E 80 (2009), Art. 066318.

[34] R. H. Kraichnan and J. R. Herring, *A strain-based Lagrangian-history turbulence theory* J. Fluid Mech. 88 (1978), $355 - 367$.

[35] P. K. Yeung, *Lagrangian characteristics of turbulence and scalar transport in direct numerical sim-

ulations, J. Fluid Mech. 427 (2001), 241 – 274.

[36] P. K. Yeung, S. B. Pope, E. A. Kurth, and A. G. Lamorgese *Lagrangian conditional statistics, acceleration and local relative motion in numerically simulated isotropic turbulence*, J. Fluid Mech. 582 (2007), 399 – 422.

[37] M. Guala, A. Liberzon, A. Tsinober, and W. Kinzelbach, *An experimental investigation on Lagrangian correlations of small-scale turbulence at low Reynolds number*, J. Fluid Mech. 574 (2007), 405 –427.

[38] J. C. R.,Hunt, A. A. Wray, & P. Moin, *Eddies, stream, and convergence zones in turbulent flows*, Center for Turbulence Research Report CTR-S88 (1988), pp. 193 — 208.

[39] K. K. Nomura & G. K. Post, *The structure and dynamics of vorticity and rate of strain in incompressible homogeneous turbulence*, J. Fluid Mech., 377 (1998), pp. 65 – 97.

[40] A. Tsinober, *An Informal Conceptual Introduction to Turbulence*, 2nd ed.,Springer, 2009.

[41] B. J. Cantwell, *Exact solution of a restricted Euler equation*, Phys. Fluids A 4 (1992), 782.

[42] P. Vieillefosse, *Local interaction between vorticity and shear in a perfect incompressible fluid*, J. Phys. (France) 43 (1982), 837.

[43] L. Chevillard and C. Meneveau, *Lagrangian dynamics and statistical geometric structure of turbulence*, Phys. Rev. Lett. 97 (2006), 174501.

[44] Y. Li, L. Chevillard, G. L. Eyink, and C. Meneveau, *Matrix exponential-based closures for the turbulent subgrid-scale stress tensor*, Phys. Rev. E, 79 (2009), 016305.

[45] L. Chevillard, L. Biferale, F. Toschi and C. Meneveau, *Modeling the pressure Hessian and viscous Laplacian in Turbulence: comparisons with DNS and implications on velocity gradient dynamics*, Phys. Fluids 20 (2008), 101504.

[46] K. Kanov, E. Perlman, R. Burns, Y. Ahmad, and A. Szalay, *I/O Streaming Evaluation of Batch Queries for Data-Intensive Computational Turbulence*, to appear *SC11*, 2011.

[47] R. J. Purser and L. M. Leslie, *An Efficient Interpolation Procedure for High-Order Three-Dimensional Semi-Lagrangian Models*, Monthly Weather Review, 119 (1991), 2492.